WL-TR-96-3055

AN EVALUATION OF COMPUTATIONAL ALGORITHMS TO INTERFACE BETWEEN CFD AND CSD METHODOLOGIES

MARILYN J. SMITH
DEWEY H. HODGES
CARLOS E.S. CESNIK

GEORGIA TECH RESEARCH INSTITUTE AND
SCHOOL OF AEROSPACE ENGINEERING
GEORGIA INSTITUTE OF TECHNOLOGY
ATLANTA GA 30332-0844

NOVEMBER 1995

FINAL REPORT FOR 09/01/94--09/01/95

Approved for public release; distribution unlimited

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7562

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Kenneth J. Moran
Aerospace Engineer
CFD Research Section

Joseph M. Manter
Chief
CFD Research Branch

Dennis Sedlock
Chief
Aeromechanics Division

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/FIMC, WPAFB, OH 45433-7913 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	NOV 1995	FINAL 09/01/94--09/01/95

4. TITLE AND SUBTITLE
AN EVALUATION OF COMPUTATIONAL ALGORITHMS TO INTERFACE BETWEEN CFD AND CSD METHODOLOGIES

5. FUNDING NUMBERS
C F33615-94-C-3010
PE 62201
PR 2404
TA 10
WU TK

6. AUTHOR(S)
MARILYN J. SMITH
DEWEY H. HODGES
CARLOS E.S. CESNIK

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
GEORGIA TECH RESEARCH INSTITUTE AND
SCHOOL OF AEROSPACE ENGINEERING
GEORGIA INSTITUTE OF TECHNOLOGY
ATLANTA GA 30332-0844

8. PERFORMING ORGANIZATION REPORT NUMBER
A-9812-100

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7562

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-96-3055

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Methods to transfer information from Computational Fluid Dynamics (CFD) to Computational Structural Dynamics (CSD) and vice versa have been evaluated. The data include geometry deformations, slopes and loads. The methods were evaluated for accuracy, robustness and case of use. Both interpolations and extrapolations were evaluated. Six methods were analyzed: Infinite Plate Splines, Multiquadrics, Thin Plate Splines, Non-Uniform B-Splines, Finite Plate Splines and Inverse Isoparametric Mapping. Of these, the Thin Plate Splines were the most accurate and robust, followed closely by Multiquadrics. The Infinite Plate Spline Method, although widely used, is not recommended.

14. SUBJECT TERMS
Computational Fluid Dynamics, Computational Aeroelasticity, Interpolation, Extrapolation.

15. NUMBER OF PAGES
470

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

FOREWORD

This report summarizes the results of an investigation into different methodologies to effect the transfer of information from a Computational Fluid Dynamics (CFD) solution grid to/from a Computational Structural Dynamics (CSD) solution grid. These data can include geometry deformations, slopes and loads. It is important that the transfer of this information provide smooth, yet accurate conversions of all types of functions that may be represented by these data (constants, linear functions, sinusoidal functions).

This research was conducted as Task 001 under the Aeromechanics Task Order Contract awarded to the Georgia Tech Research Institute in September 1994. The research was conducted from September 1994 to September 1995. This work was funded under Air Force contract F33615-94-C-3010 for the Aeronautical Systems Center, Wright-Patterson Air Force Base, Ohio. The program technical monitor was Dr. Don Kinsey. The task technical monitor was Capt. Ken Moran of the CFD Research Branch of Wright Laboratory, Wright-Patterson Air Force Base, Ohio.

The program manager for the Aeromechanics Task Order Contract at the Georgia Tech Research Institute was Mr. Robert Englar. The principal investigators for this project were Dr. Marilyn J. Smith of the Aerospace Laboratory, Acoustics and Aerodynamics Branch, Georgia Tech Research Institute and Dr. Dewey H. Hodges of the School of Aerospace Engineering, Georgia Institute of Technology. The principal investigators were assisted by Dr. Carlos E. S. Cesnik of the School of Aerospace Engineering, Georgia Institute of Technology.

1. INTRODUCTION

Over the past decade, intensive research has been invested to develop Computational Fluid Dynamics (CFD) methodologies for analysis and design of fixed wing aircraft. These methodologies range from panel methods based on the potential equations to the solution of the full time-averaged Navier-Stokes equations. Grid topologies range from regular, Cartesian structured grids to unstructured, curvilinear grids.

In most instances these CFD methodologies have become relatively mature, so that the focus of current research and utilization of these methodologies is in more complex, interdisciplinary problems. One of these interdisciplinary applications is Computational Aeroelasticity (CAE). Traditionally, CFD methods have been applied to rigid configurations. In flight, aircraft components are rarely, if ever, completely rigid. The flexibility of the structure has a direct impact on aircraft performance, loads, maneuverability and flight controls, etc. Thus, the ability of CFD methods to capture this flexibility can improve the capability of designers and analysts to understand the complex interaction of unsteady aerodynamics and structural dynamics. This understanding can ultimately lead to a reduction in production and development costs by identifying deficiencies during the design/analysis phase of development. Additionally, this capability can aid in the analysis of problems that develop in the field as the role of aircraft is redefined and expanded.

There are three primary classes of high-level dynamic computational aeroelastic methodologies, all of which can benefit from more accurate interface methodologies. The first class, and currently the most widely used, is a closely or tightly coupled aeroelastic analysis, examples of which are ENS3DAE [1], ENSAERO [2], and CFL3DAE [3]. A schematic of the typical procedure used by these methods is shown in Figure 1-1. The aerodynamic and structural dynamics modules remain independent in their solutions, and their interaction is limited to the passage of surface loads and surface deformation information.

The second class of methodologies is known as a fully-coupled analysis or unified fluid-structure interaction. This method involves the reformulation of the governing equations where both the fluid and structural equations are combined into one set of equations. These new governing equations are solved and integrated in time simultaneously. An example of this application is the research code developed by Guruswamy [4]. Each of these two methods has advantages and disadvantages, but both methods are constrained in that the grid must be updated at each iteration where deflections occur.

The development of these two classes of aeroelastic methodologies is very expensive. In addition, there has been a large investment of funds and manpower in the development of classical, rigid CFD analyses that have been tailored specifically to different applications which may require completely different methodologies to provide accurate simulation results. Moreover, the learning curve for a new CFD methodology can be prohibitive in some instances. These factors have led to the development of a third class of aeroelastic methodology, a loosely-coupled analysis. This methodology uses CFD analyses which are updated by surface deflections only after partial or full convergence of the aerodynamic loads. Thus grid deflection updates are performed sparingly, usually 3 – 10 times per analysis. Using this concept, development of a methodology which links, in general, any CFD code with any Computational Structural Dynamics (CSD) methodology is desired to accomplish a loosely-coupled aeroelastic analysis without the cost and time constraints of the detailed aeroelastic codes.

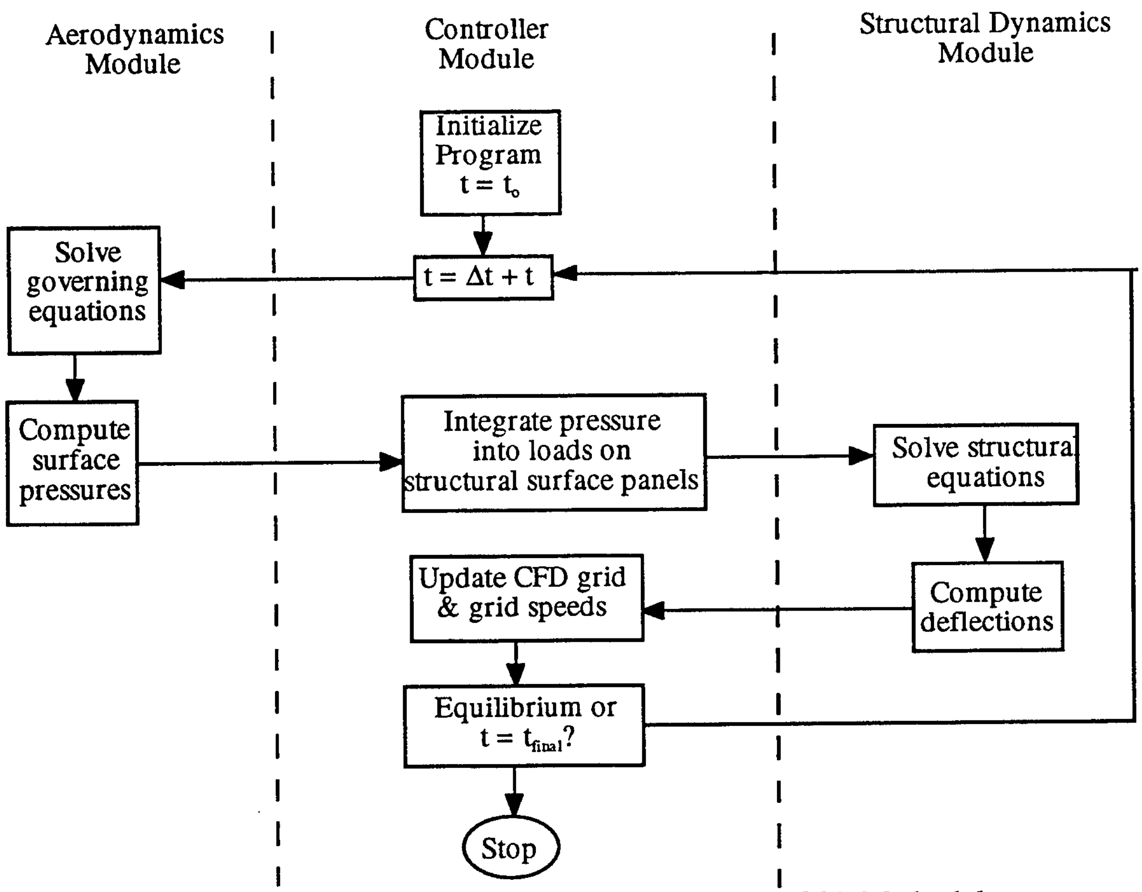


Figure 1-1. Typical Schematic for a Coupled CFD-CSD Methodology.

Although the transfer of geometry deformation data between CFD and CSD methods seems at first to be trivial, this is far from the case. The primary difficulty lies in the basic differences between the nature of the methods. CFD analyses are concerned with the flow field surrounding the surface exposed to the flow. For example, flow around a rigid airfoil is dependent only on the profile of the airfoil. The internal structure that forms the shape of the airfoil is immaterial. Conversely, CSD methods examine the airloads on the surface and how these loads affect the internal structure of the airfoil. Thus, a CFD grid is very fine around the exterior of the airfoil, wherever the changes in the flow field characteristics are expected to be a maximum. The CSD grid may be both on the surface and within the interior of the airfoil, and is oriented to the structural components. Thus, the CFD and CSD grids are not only different in grid density, but quite likely the transfer of data between the two grids requires extrapolation and interpolation.

The linear and surface splines in use today were developed for beam and plate models and are not suitable in many applications to the shell structures that are being analyzed in the current state-of-the-art aeroelastic codes. These methods may introduce oscillations, discontinuities, or poor accuracy in the surface deformations, thus producing large errors in the final solution. This is particularly true for leading and trailing edges of wings and other regions of high curvature. A systematic method is necessary to examine existing interpolation schemes, assess their strengths and weaknesses, modify them to enhance their usefulness (or develop new ones), and analytically assess their applicability for a wide range of problems.

2. APPROACH

A full review of algorithms that may be used for this application has been undertaken as part of the development of a generic interface method. In addition, the manner through which information is passed from the fluid regime to the structural regime has been examined. The top candidates have been selected using the criteria of accuracy, smoothness, ease of use, robustness, and efficiency, as shown in Figure 2-1. These candidates have undergone stringent mathematical analyses to examine their suitability in this application.

The first subtask of this research was to perform an extensive literature search. The literature search served two primary purposes: 1) to identify or eliminate possible interpolation schemes based on previous research, and 2) to aid in and reduce the amount of investigation which must be done to determine the suitability of a potential scheme. This literature search encompassed not only methods applied to CFD-CSD interpolation, but also to other engineering disciplines, as well as mathematical or scientific (physics, geology, meteorology, etc.) applications.

The selection of candidate algorithms was made on the basis of the results of the literature search, as well as the experience of the principal investigators. In addition, the selection of possible candidates was made on the basis of the criteria in Figure 2-1, and the availability of sufficiently detailed technical information for implementation.

Analytical analyses have been performed to examine the behavior of the functions in situations that may be encountered in applications and that isolate specific behaviors such as smoothness and extrapolation. In addition, the functions were analyzed for their characteristics in one-, two- and three-dimensional applications. Since these functions must provide both interpolation and extrapolation, the characteristics of their behavior limits of operation were examined. Additionally, the algorithm's behavior was assessed for both flat and highly-curved contours.

Once the algorithms were evaluated from an analytical viewpoint, they were also evaluated using a number of actual test cases under study in current research initiatives. These test cases included wings (AGARD 445, F-16), a wing-body-vertical tail with rigid body (F-16), an engine component (axisymmetric engine liner), and a lifting-body (generic hypersonic vehicle).

The report is structured to provide a concise overview of the research project. The results of the literature search are discussed in Chapter 3. The mathematical formulation of each method tested is discussed in Chapters 4 – 9. Next, the analytical test cases are described in Chapter 10, followed by a description of each method's performance in Chapter 11. In Chapter 12, the applications test cases are described, with the results discussed in Chapter 13. Chapters 14 and 15 provide conclusions and recommendations, respectively.

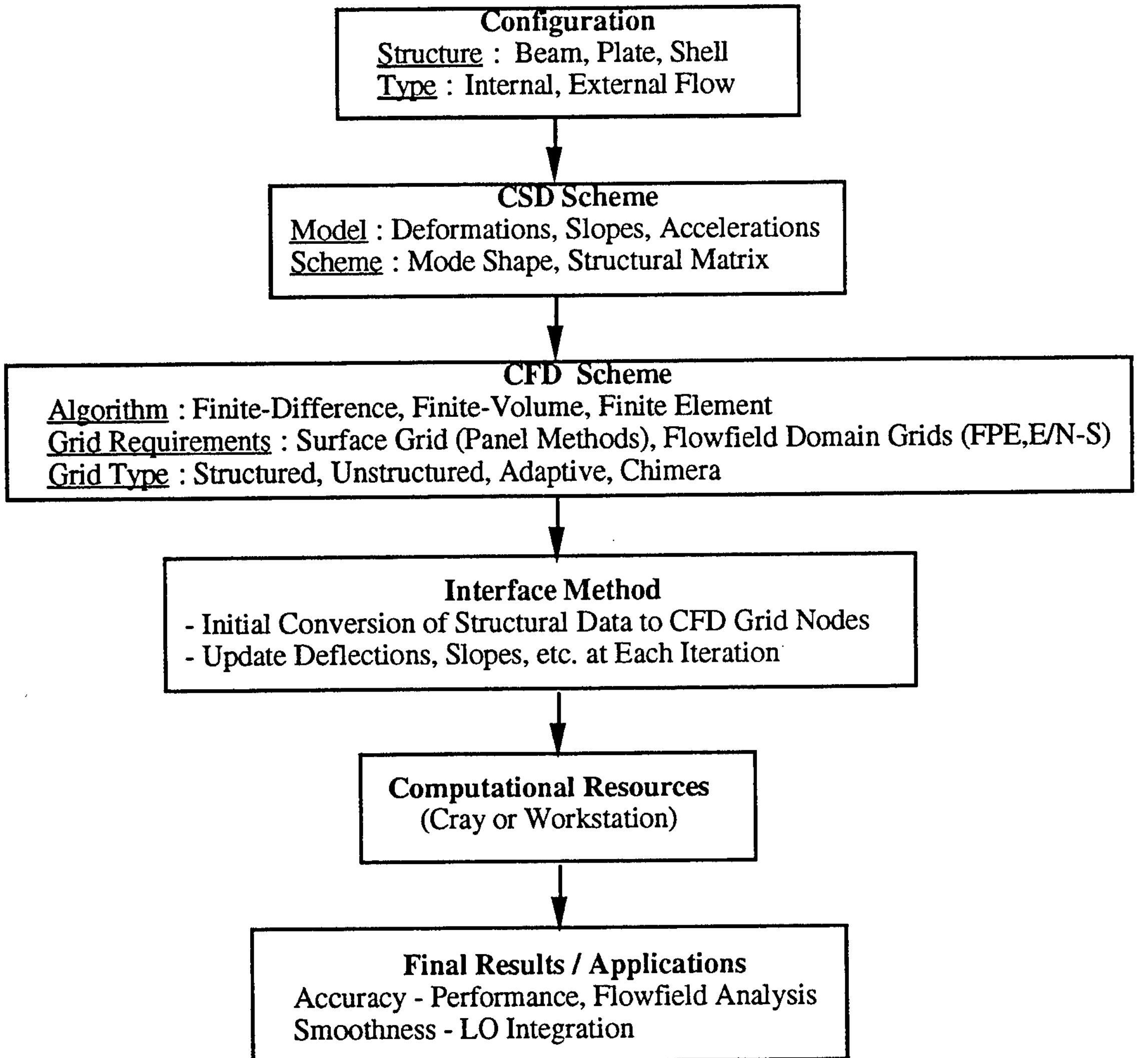


Figure 2-1. Elements to Consider when Choosing a CFD-CSD Interpolative Scheme

3. LITERATURE SURVEY

The literature survey for this research project was designed to accomplish two goals. First, it was imperative to determine what methods are currently being used for the purpose of interfacing different grid topologies and their limitations. Second, the problem of manipulation of this type of data is not exclusive to the field of CFD/CSD interfacing. Thus, it was necessary to determine what methods are being applied or have been rejected by other disciplines.

The overall literature search was initially accomplished over a two-month period, with additional research for specific information throughout the execution of this project. The initial literature search consisted of books, proceedings of conferences, journals, and government reports. A series of keywords was utilized to narrow the search to the pertinent area of interest. These keywords included:

Aeroelasticity and Computational Fluid Dynamics

Aeroelasticity and Guruswamy

Aeroelasticity and Finite Element Methods

Aeroelasticity and Interpolation

Aeroelasticity and Extrapolation

Bivariate Interpolation

Multivariate Interpolation

CFD and Extrapolation

CFD and Interpolation

CFD and Aerodynamic Loads

Polynomial Interpolation

Spline

Transfinite Interpolation

For these keywords, an initial list of 500 abstracts was extracted. Each abstract was reviewed to determine its pertinence and applicability to this research. From this list, 30 documents were obtained that were relevant to this research. From these 30 references, an additional 70 documents were reviewed. A list of these references is included in Appendix A.

The results of the literature search yielded a number of methods that appeared to be promising. These methods included: B-splines, infinite-plate splines, finite surface splines, k-vertex splines, and F-surfaces. These methods have been previously applied to aeroelastic interface problems, finite element analyses, CAD design systems or terrain mapping systems.

Each of these methods was initially screened for positive characteristics based on their mathematical derivations or their performance in direct applications. The characteristics that were reviewed include:

accuracy –	How accurate is the scheme for predicting interior data and reproducing endpoints? How sensitive is it to scaling factors?

smoothness –	Was the surface generated by the scheme smooth (C^0 continuity) or were oscillations introduced?
diminishing variation –	As the interpolated data become finer, does the scheme preserve monotonicity?
robustness –	Can the scheme handle a wide range of data interpolation ranging from flat surfaces to surfaces with high-frequency oscillations?
extrapolation –	How well does the scheme handle extrapolations?
CPU memory –	How expensive in memory is the scheme?
CPU time –	How efficient is the scheme?

The range of methods that appeared to have merit for further analysis was too large to be adequately accomplished during this research project. Thus, further reduction was required. During the 1970's and 1980's, Richard Franke investigated a number of scattered data interpolation methods for two-dimensional problems. His results [5] were very instrumental in determining which methods should be extended to three dimensions and tested further in this study. This work was primarily used to eliminate those methods identified in the literature search that proved to be less than adequate during Franke's studies. Conversely, identified methods that had excellent results reported by Franke were incorporated into this study. Specifically, the Multiquadrics method by Hardy and the Thin-Plate Spline method by Duchon were described by Franke to have excellent characteristics in two dimensions. Franke's evaluation table [5] is reproduced here as Table 3.1 for easy reference.

In addition to the literature search, a survey was sent out by Mr. Larry Huttsell of Wright Laboratory (FIB) to determine what methods are currently in use for aeroelastic interfaces, which may not be available in technical journals or reports. The survey form is included as Figure 3-1. The results of the survey are tabulated in Table 3.2. The survey yielded thirteen responses from the United States. The primary aeroelastic methods in use in industry appear to be those which rely on lower-order aerodynamic methods, including panel methods and lifting-line and lifting-surface methods. The aeroelastic option in MSC/NASTRAN has a number of industry users. Euler and Navier-Stokes methods appear to still be in development, as the only sources of these are government agencies or universities under contract to government agencies. The primary method of interfacing between the structural and aerodynamic grids is the Infinite-Plate Spline of Harder and Desmarais, as implemented in MSC/NASTRAN and a number of independent codes. The next most popular method is the use of shape functions in beam and plate representations. A number of researchers responded that the accuracy of their method could not be readily determined because of the lack of good graphics. Extrapolation was also mentioned as a problem by several engineers. The results of this survey were used to help choose methods that appeared to work well (shape functions), as well as to analyze methods which are in use, but have significant limitations (Infinite-Plate Spline).

As a result of this literature survey, the following methods were chosen for further analysis:

- Infinite-Plate Spline by Harder and Desmarais – now in use in many aeroelastic methods
- Finite-Plate Spline by Appa – now in use in several aeroelastic methods
- Biharmonic-Multiquadric by Hardy – rated excellent by Franke
- Thin-Plate Splines by Duchon – rated excellent by Franke
- Inverse Isoparametric Mapping – a shape function method with some promising results (at the commencement of this research, this method was under development at WL)
- Non-Uniform B-Splines – a CAD package implementation (the DT_NURBS library was recommended by WL)

These methods were analyzed by a series of mathematical test cases and selected applications. The results of these tests are discussed in the remaining sections of this report.

Table 3.1 Evaluation of Interpolation Methods by Franke

Program Number	Description	Global/Local Type	Continuity (Continuous Derivatives	Precision (Polynomial)	Storage	Domain	Sensitivity	Complexity	Accuracy	Visual	Timing (Eval/100 point total)
1	Franke's Method - 3	L2	2	1	=11N	R^2	C	D	B	B	C/B
4	Akina's Method	L4	1	1	<33N	R^2	C	D	B	C	A/A
10	Akina's Method - Mod I	L4	1	1	<33N	R^2	C	D	B	C	A/A
13	Nielson-Franke Quad.	L3	1	2[a]	<32N	R^2	B	D	A^-	A^-	A/B^-
14	Mod. Quad. Shepard	L1	1	2[a]	5N	B[d]	B	B	A^-	A^-	C/D
16	Akina's Method - Mod III	L4	1	2[a]	<33N	R^2	B	D	B^+	B^+	A/B^-
24	Franke's Method - TPS	L2	1	1	=13N	R^2	C	D	B^+	B^+	B/B
28	Lawson's Method	L4	1	2[a]	<18N	$CH((x_k, y_k))^G$	N[g]	D	B	B	A/A
19	Nielson's Min Norm Net.	G4	1	1	<32N	$CH((x_k, y_k))^G$	N	D	A^-	A^-	B/B
21	Hardy's Multiquadrics	G6	-	-1[e]	$\frac{1}{2}N(N+4)$	R^2	B	A	A	A	B^-/C^-
23	Duchon's Thin Plate Splines	G6	1[b]	1	$\frac{1}{2}(N+3)(N+7)$	R^2	N	A	A	A	C/D
27	Hardy's Recip. Multiquadric	G6	-	-1	$\frac{1}{2}N(N+4)$	R^2	B	A	A	A	B^-/C^-
30	Foley's TFΔ Cubin Spline	G5	2	-1	=8N	R^2	C	C	B^+	B	B/D
2	Mod. Shepard • Plane	L1	1	1	0	B	C	B	C	C	D/D
3	Mod. Linear Shepard	L1	1	1	2N	B	C	B	C	C	C/C
5	McLain's Method M_{10}	G1	-	1	0	R^2	C	B	B^+	B^+	F/F
6	Franke's Method - 1	L2	1	-1	=10N	R^2	C	D	B	B	C/B
7	Mod. Shepard's Method	L1	1	0	0	B	C	A	D	D	B^-/B^-

[a] except for certain possible dispositions of points
[b] C^∞ except at data points
[c] convex hull of $\{(x_k, y_k)\}$
[d] $B = \{(x, y)\}: d_k(x, y) < R_{w1}$, some K)
[e] no polynomial precision
[f] program modified by this investigator to extrapolate to all of R^2
[g] no parameter
[h] extrapolates beyond convex null, not all of R^2
[i] estimated (360/67 no longer available)
[j] these methods tested since the appearance of [18]
[k] at least a rectangle given by user

Table 3.1 Evaluation of Interpolation Methods by Franke (Concluded)

Program Number	Description	Global/Local Type	Continuity (Continuous Derivatives	Precision (Polynomial)	Storage	Domain	Sensitivity	Complexity	Accuracy	Visual	Timing (Eval/100 point total)
8	Mod. McLain's M_8	L1	1	1	0	B	C	B	C	C^-	C/C
11	Akina's Method - Mod II	L4	1	1	<33N	R^2	C	D	B	C	A/A
12	Nielson-Franke Linear	L3	1	1	<32N	R^2	C	D	C	C	A/A
17	Quad. Shepard's Method	G1	-	2	5N	R^2	N	B	F	F	D/F
18	Shepard's Method	G1	-	0	0	R^2	N	A	F	F	C/C
20	Rotated Gaussians	G6	-	-1	$\frac{1}{2}N(N+4)$	R^2	D	A	B^+	B^+	B^-/C^-
22	Duchon's Radial Cubic	G6	2^b	1	$\frac{1}{2}(N+3)(N+7)$	R^2	N	A	A	A	C/D
25	Foley's Generalized Newton	G5	-	-1	=8N	R^2	N	C	B^-	B	B^-/C
26	Foley's TFΔ Bernstein	G5	-	-1	=8N	R^2	B	C	B	B	D/C^-
29	Rotated B-Splines	G6	2	-1	$\frac{1}{2}N(N+4)$	R^2(=0 outside B)	D	A	B^+	B	D/D^-
31	Foley's Shep. Δ Cub. Spl.	G5	2	0	=8N	R^2	C	C	B	B	B/C
32[j]	Akina's Method - Mod 80	L4	1	1	=33N	R^2	N	D	C^+	C^-	A/A
33[j]	Little's Method	L4	1	1	=15N	H^h	N	D	A^-	A^-	B/B^i
34[j]	Vittitow's Adaptive Maude	L7	1	2	=27N	$=G^k$	C	D	C	C	B/B^i

[a] except for certain possible dispositions of points
[b] c^∞ except at data points
[c] convex hull of $\{(x_k, y_k)\}$
[d] $B = \{(x, y)\}: d_k(x, y) < R_{w1}$, some K)
[e] no polynomial precision
[f] program modified by this investigator to extrapolate to all of R^2
[g] no parameter
[h] extrapolates beyond convex hull, not all of R^2
[i] estimated (360/67 no longer available)
[j] these methods tested since the appearance of [18]
[k] at least a rectangle given by user

SURVEY

We are surveying aeroelasticity researchers to determine what kinds of interface methodologies are used to transfer information between the structural and computational grids. Please take a moment to respond to this survey. Thank you.

1. What interface method do you use to convert between the structural grid and the computational grid in your aeroelastic simulation method?

2. What types of structure do you simulate:

____ lifting surfaces (plate with bending)
____ lifting surfaces (shells for wings, canards, etc.)
____ lifting bodies (shells for fuselages, engines, etc.)
____ other: (Please specify) ____

3. What problems or limitations have you incurred using this (these) methods?

4. What Aeroelastic Methodology are you using?______________________
(If the code name is not widely know, we would appreciate any papers you may have published to that we may correctly acknowledge your work.)

5. Your name ____________________
Position____________________
Company/Country____________________
e-mail or other contact address________________________________

Figure 3-1. Survey Letter Which was Sent to US Industry and Government Facilities.

Table 3.2 Compilation of the Aeroelastic Survey

Company/ Organiz.	Aeroelastic Methodology in Use	Lifting Surfaces *	Lifting Surfaces **	Lifting Bodies +	Other	What Interface Methods are in Use	What Problems Have Been Encountered with the Interface Methodology
NASA-DFRC	STARS (NASA TM 101703 & TM 4544	X	X	X		Interpolation (Method not specified); Use Common FE CFD/CSD Methods	None
NASA-ARC	ENSAERO	X	X		Wing-Box	Virtual Surface Method based on Consistent Load Approach; Interpolation based on Shape Functions	None, if patched structured grids are used
NASA-LaRC	ISAC, FAST, NASTRAN, CAP-TSD, CFL3DAE	X	X	X		Harder & Desmarais Surface Splines. Includes Smoothing, Amplitude and Rotations are Independently Fitted, Discontinuous Regions are Permitted	Interpolation can sag between points and thus requires care in the selection of input points, scaling and graphical monitoring of results. Must not extrapolate to any significant degree. Requires sign. labor to interface with FEM models by choosing input points from FEM, dividing regions, deleting close x-y points, etc. Modern graphical interfaces are not well developed.
Gulfstream Aerospace	MSC/ NASTRAN with Aero Corrections	X (Wind Tunnel Models)			Wing-body-em-pennage using beam models	Linear Splines (within MSC/NASTRAN)	Lack of an aerodynamic factoring scheme
Boeing	ELFINI		X	X		Monomial Shape functions related to str. displacements by least squares. Unit loading functions from principle of virtual work convert cps to loads.	No limitations, although care must be taken in setting up the functions to accurately interpolate

Table 3.2 Compilation of the Aeroelastic Survey (cont.)

Company/ Organiz.	Aeroelastic Methodology in Use	Lifting Surfaces *	Lifting Surfaces **	Lifting Bodies +	Other	What Interface Methods are in Use	What Problems Have Been Encountered with the Interface Methodology
Lockheed-Martin, Fort Worth Co.	Cunningham Kernel Function, Doublet Lattice, Zona6, Zona7	X	X	X	Beam Models	Surface Splines	If a fuselage is represented as a simple beam, "invented" structure must be included in model to extend to lifting surfaces. Local deformations are a problem with built-up fuselages, in particular.
MacNeal-Schwendler Corp.	MSC/ NASTRAN				Finite Element Models	Infinite-Plate Spline of Harder and Desmarais and linear bending/twisting spline with rigid offsets	None with linear spline; curling up of extrapolated regions on IPS
Structural Dyanmics Research Corporation	MSC/ NASTRAN, Doublet Lattice, Mach Box, Piston Theory	X	X	X	Bars along elastic axis	Same as MacNeal-Schwendler Entry	Poor graphics gives lack of visibility of accuracies
Southwest Research Institute	ASTROS, TSO	X		X	Internal flow flat panels (ducts)	Cubic Spline	None
MDA	CAP-TSD, NASTD	X	X	X		Equivalent virtual work force mapping.	Iterative solution takes too long
ZONA Technology , Inc.	ZONA codes (panel methods)	X	X	X		Line and Surface Interpolation	Accuracy Decreases for Higher Order Modes
Dynamic Engineering , Inc.	MSC/ NASTRAN	X			Bar elements	Same as MacNeal-Schwendler Entry	Aerodynamic elements are not comp. with post-processing, numbering scheme limitations, problems with element alignment
Georgia Tech Research Institute	ENS3DAE	X	X	X		Spline Interpolations, Infinite-Plate Splines	Accuracy at higher modes is degraded, Oscillations are introduced between nodes, Multiple runs to optimize interpolations

* Plates with Bending ** Shells with Bending + Shells for Fuselages, Engines, etc.

4. INFINITE-PLATE SPLINE METHOD – MATHEMATICAL FORMULATION

Nomenclature

∇^4	Biharmonic operator
W	Plate deflection
D	Plate flexural rigidity
q	Distributed load on the plate
P	Point load
F_i, F_{a_i}, F_{s_i}	Normalized point loads
A,B,A_i,B_i	Undetermined coefficients
x,y	Coordinates of a generic point (Cartesian)
r,θ	Coordinates of a generic point (polar)
x_i, y_i	Known grid point coordinates
x_{a_i}, y_{a_i}	Known aerodynamic grid point coordinates
x_{s_i}, y_{s_i}	Known structural grid point coordinates
$\{F\}$	Column matrix of normalized forces F_i
$\{W_s\}$	Column matrix of deflections for the structural grid
$[C_s]$	Structural influence matrix
$\{W_a\}$	Column matrix of deflections for the aerodynamic grid
$[C_a]$	Aerodynamic influence matrix
$\{F_a\}$	Column matrix of normalized aerodynamic forces F_{a_i}
$\{F_s\}$	Column matrix of normalized structural forces F_{s_i}
$[C_{sa}]$	Matrix of flexibility influence coefficients relating structural and aerodynamic grid
$[I_{as}]$	Deflection interpolation matrix
$[I]$	Identity matrix

Definition

The method of infinite-plate splines (IPS) is one of the more popular methods of interpolation, being used in programs such as ASTROS and MSC/NASTRAN. This method is based on a superposition of the solutions for the partial differential equation of equilibrium for an infinite plate. We first consider a set of N discrete "grid points" (x_i, y_i), for $i=1, 2, \ldots, N$, lying within a two-dimensional domain with Cartesian coordinates x and y. Each grid point has associated with it a "deflection" $W_i(x_i, y_i)$ that defines the vertical position coordinate of the surface on which both structural and aerodynamic grid points are presumed to lie. Using solutions of the equilibrium equation for an

infinite plate, one calculates the values for a set of concentrated loads, all presumed to act at the known data points, that give rise to the required deflections $W_i(x_i, y_i)$. Those concentrated forces are then substituted back into the solution, thus providing a smooth surface that passes through the data. Thus, given the deflections of the structural grid points it is possible to interpolate to a set of aerodynamic grid points that, in general, do not coincide with the structural ones. Some advantages to this method are that the grid is not restricted to a rectangular array and that the interpolated function is differentiable everywhere. Points far away from known points are extrapolated nearly linearly. The method described here was developed by Turner [1] who expanded the work of Harder and Desmarais [2] into a working FORTRAN code. For IPS interpolation a minimum of three points is required, since three points define a plane. Methods of interpolation are classified in two ways: global interpolation, in which the entire surface is fitted with a set of functions, and local interpolation, in which only a subset of the surface is fitted with appropriate functions; see Rodden [3]. Although Turner's method uses local interpolation, the present implementation is global.

Current Uses in Aeroelastic Applications

Application Name	Comments	Code Owner
NASTRAN	uses combination of linear and infinite-plate surface splines with beam and plate elements	MacNeal-Schwendler Corp.
XTRANS3S	uses MPROC to provide infinite-plate spline interpolation	Wright Laboratory/FIB
ASTROS	same as in NASTRAN	Wright Laboratory/FIB
ENS3DAE	uses infinite-plate spline in 1-D flexible applications; uses MPROC3D [4] (infinite-plate splines with smoothing options and singular value decomposition) for 2-D/3-D flexible applications	Wright Laboratory/FIB
CFL3DAE	uses MPROC to provide infinite-plate spline interpolation (MPROC is a planar version of MPROC3D; MPROC, developed at NASA Langley, does not contain smoothing and singular value decomposition of MPROC3D.)	NASA Langley

Mathematical Description

In this section the general equations developed by Harder and Desmarais [2], based on small deflections of an infinite plate, will be discussed. The plate extends to infinity in both directions, deforms only in the direction normal to the plate, and has bending stiffness only. The governing equation is

$$D\nabla^4 W = q \tag{1}$$

where W is the plate deflection, D is the plate flexural rigidity, and q is the distributed load on the plate. Introducing polar coordinates, $x = r\cos\theta$, $y = r\sin\theta$, so that ∇^4 in polar coordinates is given by [5]

$$\nabla^4 = \nabla^2\nabla^2 = \left(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\right)\left(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\right) \tag{2}$$

and considering the deflection due to a point load P at the origin of the coordinate system, a general solution of equation (1) can be written as

$$W(r) = A + Br^2 + \left(\frac{P}{16\pi D}\right) r^2 \ln r^2 \tag{3}$$

Here A and B are undetermined coefficients.

We now consider superposition of N point loads at various given locations (x_i, y_i), for i=1, 2, ..., N in the 2-D space. To accomplish this, a simple shift in the coordinate system is necessary. Thus, the deflection due to the N point loads, evaluated at any point (x, y) in the 2-D space, can be determined by simply replacing r^2 in equation (3) by $r_i^2 = (x - x_i)^2 + (y - y_i)^2$ and summing over the points. Therefore,

$$W(x, y) = \sum_{i=1}^{N} \left(A_i + B_i r_i^2 + F_i r_i^2 \ln r_i^2\right) \tag{4}$$

where A_i, B_i, and $F_i = \frac{P_i}{16\pi D}$ are undetermined coefficients.

For the purpose of determining these undetermined coefficients one needs to use certain information about the solution. Harder and Desmarais [2] showed that by expanding equation (4) for large values of r, one obtains terms of order r^2, r, 1, $1/r$, etc., along with terms of order $r^2\ln r^2$, $r\ln r^2$, $\ln r^2$, etc.

$$\begin{aligned} W(r,\theta) = {} & r^2 \ln r^2 \sum_{i=1}^{N} F_i + r^2 \sum_{i=1}^{N} B_i - \\ & 2r\ln r^2 \sum_{i=1}^{N} (x_i\cos\theta + y_i \sin\theta) F_i - \\ & 2r \sum_{i=1}^{N} (x_i\cos\theta + y_i\sin\theta)(F_i + B_i) + \\ & \ln r^2 \sum_{i=1}^{N} \left(x_i^2 + y_i^2\right) F_i + \ldots \end{aligned} \tag{5}$$

To ensure that the solution does not become excessively large as r becomes large, one needs to suppress some of these terms; viz., coefficients of the terms of order r^2, $r^2\ln r^2$, and $r\ln r^2$ must be zero. It is evident that this can be accomplished by setting

$$\sum_{i=1}^{N} F_i = 0 \tag{6}$$

$$\sum_{i=1}^{N} x_i F_i = 0 \tag{7}$$

$$\sum_{i=1}^{N} y_i F_i = 0 \tag{8}$$

$$\sum_{i=1}^{N} B_i = 0 \tag{9}$$

Here equation (6) can be recognized as the discrete force equilibrium equation, which eliminates terms of order $r^2 \ln r^2$. Equations (7) and (8) are discrete moment equilibrium equations and eliminate terms of order $r \ln r^2$. Finally, equation (9), the physical significance of which is not clear, serves to eliminate terms of order r^2.

Therefore, returning to the exact version, equation (4), it is clear that one can express W as

$$W(x, y) = a_0 + a_1 x + a_2 y + \sum_{i=1}^{N} F_i r_i^2 \ln r_i^2 \tag{10}$$

where a_0, a_1, and a_2 are unknowns given by

$$a_0 = \sum_{i=1}^{N} \left[A_i + B_i \left(x_i^2 + y_i^2\right)\right] \tag{11}$$

$$a_1 = -2 \sum_{i=1}^{N} B_i x_i \tag{12}$$

$$a_2 = -2 \sum_{i=1}^{N} B_i y_i \tag{13}$$

Note that the $N+3$ unknowns in equation (10) can be determined from application of side conditions found in equations (6) – (8) along with setting the deflection at the i^{th} point to its known value W_i. Viz.,

$$W_i = a_0 + a_1 x_i + a_2 y_i + \sum_{j=1}^{N} r_{ij}^2 \ln r_{ij}^2 F_j \quad \text{for } i=1, 2, \ldots, N \tag{14}$$

where

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 \tag{15}$$

is the square of the distance between known points (x_i, y_i) and (x_j, y_j).

Equation (14) and the side conditions found in equations (6) – (8) can now be expressed in matrix form as

$$\{W\} = [R]\{a\} + [Z]\{F\} \tag{16}$$

and

$$[R]^T \{F\} = 0 \tag{17}$$

where

$$\{W\} = \begin{Bmatrix} W_1 \\ W_2 \\ \vdots \\ W_N \end{Bmatrix}, \quad \{F\} = \begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_N \end{Bmatrix}$$
$$\{a\} = \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix}, \quad [R] = \begin{bmatrix} \begin{Bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{Bmatrix} & \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{Bmatrix} & \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{Bmatrix} \end{bmatrix} \tag{18}$$

and where $[Z]$ contains the elements Z_{ij} given by

$$Z_{ij} = r_{ij}^2 \ln r_{ij}^2 \tag{19}$$

Thus, the deflection W_i, for i=1, 2, ..., N, can be determined using equations (16) and (17) along with the above definitions.

Implementation

In order to make use of equations (16) and (17) to create a transformation matrix for interpolation, we follow the works of Schmitt [6] and Rodden [3].

Rodden [3] relates a known matrix $[C_s]$ to the deflections and forces of the structural grid as

$$\{W_s\} = [C_s]\{F_s\} \tag{20}$$

where $\{W_s\}$ is a column matrix, the elements of which are N_s deflections of the structural grid $W_{s_i}(x_{s_i}, y_{s_i})$, i=1, 2, ..., N_s, $\{F_s\}$ is a column matrix containing the unknown normalized forces F_{s_i}, i=1, 2, ..., N_s, and $[C_s]$ is an unknown $N_s \times N_s$ matrix.

It now remains for the coordinates of the aerodynamic grid points $(x_{a_i}, y_{a_i}, W_{a_i})$, i=1, 2, ..., N_a to be found. An expression similar to equation (20) exists relating the deflections and forces of the aerodynamic grid so that

$$\{W_a\} = [C_a]\{F_a\} \tag{21}$$

Here the $N_a \times 1$ column matrices $\{W_a\}$ and $\{F_a\}$ and the $N_a \times N_a$ matrix $[C_a]$ are analogous matrices for the N_a aerodynamic grid points. The above relationship is made equivalent to the structural relationship of equation (20) that $\{F_a\}$ will produce the same deflection $\{W_s\}$ as $\{F_s\}$. Thus there will exist some $N_s \times N_a$ matrix $[C_{sa}]$ such that

$$\{W_s\} = [C_{sa}]\{F_a\} \tag{22}$$

Maxwell's law of reciprocity, as defined by Meirovitch [5] for example, requires the existence of a reciprocal relation that relates $\{W_a\}$ to $\{F_s\}$ of the form

$$\{W_a\} = [C_{as}]\{F_s\} \tag{23}$$

where $[C_{as}]$ is the transpose of $[C_{sa}]$. Substituting equation (20) into equation (23) and given that $[C_s]$ is symmetric, one obtains

$$\{W_a\} = [C_{as}]\,[C_s]^{-1}\,\{W_s\} \tag{24}$$

Substituting equations (21) and (22) into equation (24), one finds that

$$[C_a] = [C_{as}]\,[C_s]^{-1}\,[C_{sa}] \tag{25}$$

A matrix $[I_{as}]$, also exists such that the interpolation

$$\{W_a\} = [I_{as}]\,\{W_s\} \tag{26}$$

can be made. Comparing equations (24) and (26) and noting that matrix $[C_s]$ is symmetric, one can identify $[I_{as}]$ and also solve for $[C_{sa}]$ in terms of as $[I_{as}]$ so that

$$[I_{as}] = [C_{as}]\,[C_s]^{-1} \tag{27}$$

$$[C_{sa}] = [C_s]\,[I_{sa}] \tag{28}$$

Substituting $[C_{sa}]$ from equation (28) and its transpose into equation (25), one can construct the transformation

$$[C_a] = [I_{as}]\,[C_s]\,[I_{sa}] \tag{29}$$

The matrix $[I_{as}]$ is now obtained from applying equation (16) to both structural and aerodynamic grids. Assuming the structural grid to be known so that the forces $\{F_s\}$ are determined from it, one finds

$$\{W_s\} = [R_s]\,\{a\} + [Z_s]\,\{F_s\} \tag{30}$$

$$\{W_a\} = [R_a]\,\{a\} + [Z_{as}]\,\{F_s\} \tag{31}$$

where, according to equation (17), equation (30) is subject to

$$[R_s]^T\,\{F_s\} = 0 \tag{32}$$

and where

$$\begin{aligned} [R_s] &= \left[\begin{Bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{Bmatrix} \begin{Bmatrix} x_{s_1} \\ x_{s_2} \\ \vdots \\ x_{s_N} \end{Bmatrix} \begin{Bmatrix} y_{s_1} \\ y_{s_2} \\ \vdots \\ y_{s_N} \end{Bmatrix} \right] \\ [R_a] &= \left[\begin{Bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{Bmatrix} \begin{Bmatrix} x_{a_1} \\ x_{a_2} \\ \vdots \\ x_{a_N} \end{Bmatrix} \begin{Bmatrix} y_{a_1} \\ y_{a_2} \\ \vdots \\ y_{a_N} \end{Bmatrix} \right] \end{aligned} \tag{33}$$

and where $[Z_s]$ and $[Z_{as}]$ contain the elements

$$
\begin{aligned}
Z_{s_{ij}} &= \left[\left(x_{s_i} - x_{s_j}\right)^2 + \left(y_{s_i} - y_{s_j}\right)^2\right] \ln\left[\left(x_{s_i} - x_{s_j}\right)^2 + \left(y_{s_i} - y_{s_j}\right)^2\right] \\
Z_{as_{ij}} &= \left[\left(x_{a_i} - x_{s_j}\right)^2 + \left(y_{a_i} - y_{s_j}\right)^2\right] \ln\left[\left(x_{a_i} - x_{s_j}\right)^2 + \left(y_{a_i} - y_{s_j}\right)^2\right]
\end{aligned} \tag{34}
$$

The bracketed expressions in these terms relate to distances between pairs of structural grid points in $[Z_s]$ and between individual aerodynamic and structural grid points for $[Z_{as}]$. Equation (30) can be solved for $\{F_s\}$ yielding

$$\{F_s\} = [Z_s]^{-1}\{\{W_s\} - [R_s]\{a\}\} \tag{35}$$

Substituting equation (35) into (32) and rearranging, one finds the undetermined coefficients as

$$\{a\} = [Y_s]^{-1}[R_s]^T[Z_s]^{-1}\{W_s\} \tag{36}$$

where

$$[Y_s] = [R_s]^T[Z_s]^{-1}[R_s] \tag{37}$$

Next, substitution of equation (36) back into (35) yields a linear relationship between $\{F_s\}$ and $\{W_s\}$, viz.,

$$\{F_s\} = \left[[I] - [Z_s]^{-1}[R_s][Y_s]^{-1}[R_s]^T\right][Z_s]^{-1}\{W_s\} \tag{38}$$

Thus, given the coordinates of the deformed structural grid $(x_{s_i}, y_{s_i}, W_{s_i})$, one determines the F_{s_i} from equation (38).

It now remains for the coordinates of the aerodynamic grid points $(x_{a_i}, y_{a_i}, W_{a_i})$ to be found. To finish this we simply substitute the results of equations (36) and (38) into equation (31) yielding

$$\{W_a\} = \left[[R_a][Y_s]^{-1}[R_s]^T + [Z_{as}]\left[[I] - [Z_s]^{-1}[R_s][Y_s]^{-1}[R_s]^T\right]\right][Z_s]^{-1}\{W_s\} \tag{39}$$

from which the transformation matrix $[I_{as}]$ (see equation 26) can be identified as

$$[I_{as}] = \left[[R_a][Y_s]^{-1}[R_s]^T + [Z_{as}]\left[[I] - [Z_s]^{-1}[R_s][Y_s]^{-1}[R_s]^T\right]\right][Z_s]^{-1}\{W_s\} \tag{40}$$

Equation (40) gives the transformation matrix $[I_{as}]$, which is used to relate deflections of the structural grid to deflections of the aerodynamic grid. The flexibility influence coefficients can also be used to relate the loads in aerodynamic grid to the deflections in structural grid. To do this, one first solves equation (20) for $\{F_s\}$ in terms of $\{W_s\}$, so that

$$\{F_s\} = [C_s]^{-1}\{W_s\} \tag{41}$$

Then, one substitutes equation (22) for $\{W_s\}$, and finally makes use of equation (28), yielding

$$\{F_s\} = [I_{sa}]\{F_a\} \tag{42}$$

Note that the influence coefficient matrix $[C_s]$ for the structural grid can be found by first comparing equations (41) and (38), identifying

$$[C_s] = [Z_s]\left[[I] - [Z_s]^{-1}[R_s][Y_s]^{-1}[R_s]^T\right]^{-1} \tag{43}$$

The influence coefficient matrix $[C_a]$ for the aerodynamic grid can then be found from equation (29). The procedure is essentially the same regardless of whether mode shape information is used or structural influence coefficient data.

The implementation of the IPS method into a working FORTRAN code is straightforward and may account for some of the success of this method. The mathematical implementation in FASIT, described here, applied required roughly 400 lines of code. The method has some very desirable features: it is straightforward, it gives a differentiable function everywhere, and it is not restricted to a rectangular grid.

Limitations

1. It was found that as the number of points used in the interpolation increases, so does round-off error, effectively negating the increased accuracy obtained from the larger number of points. Turner [1] has tested interpolations using 3 – 10 points and recommends 7 points for optimum results.

2. The interpolation method presented here is 2-D. For a wing, this assumes, among other things, that the planform is flat. It has difficulty with deflections caused by deformations other than simple "out-of-plane" (plate) bending/twisting. For example, in the application MPROC3D [4], it was noted that the IPS method has difficulties with in-plane mode shapes that are very small. This reduces its usefulness in aeroelastic applications. Thus, in the form implemented here (a global interpolation), use of the IPS method should be limited to out-of-plane bending/twisting only. (It should be noted, however, that extension to 3-D and to a local method is straightforward; see the section on Thin-Plate Splines. When used as a local method, IPS is limited to approximate C^0 continuity between adjoining subdomains.)

3. IPS creates distortions or oscillations in extrapolated regions (e.g., wing tips). This has been described as a curling up or "potato chip" effect.

4. The IPS method has a tendency to introduce high-order oscillations for some rapidly varying functions. These can be eliminated by smoothing techniques at the expense of accuracy. The smoothing must be made on a curve-by-curve basis and is very time-consuming.

References

[1] Turner, E. W., "Interpolation of Flexibility Influence Coefficients from one Grid to Another Using Infinite Surface Splines," Technical Memorandum WRDC-TM-91-000-FIBE, Wright Laboratories, Wright-Patterson Air Force Base, Ohio, July 1991.

[2] Harder, R. L. and Desmarais, R. N., "Interpolation Using Surface Splines," *Journal of Aircraft*, Vol. 9, No. 2, 1972, pp. 189 – 191.

[3] Rodden, William P., "Further Remarks On Matrix Interpolation of Flexibility Influence Coefficients," *Journal of Aerospace Sciences*, Vol. 26, November 1959, pp. 760 – 761.

[4] Shan, R., "MPROC3D: User's and Program Reference Guide," Unpublished report, 1993.

[5] Meirovitch, L., *Analytical Methods in Vibrations*, MacMillian Publishing Co., New York, 1967.

[6] Schmitt, Alfred F., "A Least Squares Matrix Interpolation of Flexibility Influence Coefficients," *Journal of Aeronautical Sciences*, October 1956, page 980.

5. MULTIQUADRIC-BIHARMONIC METHOD – MATHEMATICAL FORMULATION

Nomenclature

α_i	Undetermined coefficients
$\{\alpha\}$	Column matrix of undetermined coefficients
$[B]$, $[B_G]$	Matrix constructed from basis functions evaluated at given points
$[B_{IG}]$	Coupling matrix constructed from basis functions evaluated at given and interpolated points
$H(\mathbf{x})$	Function to be interpolated
$\{H\}$, $\{H_G\}$	Column matrices of given values of the function to be interpolated
$\{H_I\}$	Column matrix of interpolated values of the function
n	Total number of points to be interpolated
N	Total number of given points
r	User defined input parameter
$\{U\}$, $\{U_G\}$	Column matrices of N 1's
$\{U_I\}$	Column matrix of n 1's
$\mathbf{x}$	Position vector to an arbitrary point in Cartesian space
$\mathbf{x}_i$, $\mathbf{x}_{G_i}$	Position vector to the ith given points in Cartesian space
$\mathbf{x}_{I_i}$	Position vector to the ith point to be interpolated in Cartesian space

Definition

The multiquadrics (MQ) method is an interpolation technique that represents an irregular surface in terms of a set of quadratic basis functions. More recently named the multiquadric-biharmonic method, it was developed to perform interpolation of various topographies [1]. The original name reflects the method's use of quadratic basis functions; note that a "quadric" surface is one whose geometry is described by quadratic equations. The quadric surface used in most cases is a circular hyperboloid in two sheets. The addition of "biharmonic" to the name is due to an important proof that the equations governing the method can always be solved [2] showing that the method is analogous to a numerical solution of Poisson's equation. The MQ method is stable and consistent with respect to a user-defined parameter r that controls the shape of the basis functions. A large r gives a flat sheet-like function, while a small r gives a narrow cone-like function. For non-zero values of r the MQ method produces an infinitely differentiable function that preserves monotonicity and convexity. The MQ method was originally implemented as a global basis function method with

constant values of r. However, later work by Kansa [3, 4] suggests that the method's conditioning, accuracy, and general numerical performance are improved by (1) permitting r to vary among the basis functions; (2) scaling and/or rotating the independent variables; and (3) applying it in overlapping subdomains.

Current Applications

Review papers by Franke [2] and Hardy [5] indicate that the method has been widely used and that, in comparison with several other interpolation methods, it performs well. It was employed to interpolate topographical data, to prepare a geoid contour map, to interpolate rainfall data in hydrology, and to remote sensing, all with significant success. Hardy's review paper [5] also alludes to applications to the areas of photogrammetry, surveying and mapping, geophysics, hydrography, and computational fluid dynamics (CFD). Kansa [3] successfully applied the MQ method in several test cases, such as interpolation from a coarse grid to a fine grid. Although the MQ method has not been directly applied in aeroelastic methodologies, Kansa's CFD applications [3, 4] interpolated data between different grids with varying degrees of success.

Mathematical Description

The MQ method is the summation of quadric surface equations with unknown coefficients to represent irregular surfaces. A quadric surface is a surface which can be mathematically defined by quadratic equations. This method was originally developed by Hardy [1] to construct a function $H(\mathbf{x})$ given its values at a set of N discrete "nodal" values $H_i = H(\mathbf{x}_i)$ at "nodes" $\mathbf{x} = \mathbf{x}_i$, for $i = 1, 2, \ldots, N$. Here $\mathbf{x} = x\check{\mathbf{i}} + y\check{\mathbf{j}} + z\check{\mathbf{k}}$; x, y, and z are Cartesian coordinates; and $\check{\mathbf{i}}$, $\check{\mathbf{j}}$, and $\check{\mathbf{k}}$ are unit vectors in the directions of x, y, and z, respectively. (Note that by omission of the z-direction the three-dimensional scheme reduces to a two-dimensional one.)

The interpolation equation was originally given as

$$H(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \left|\mathbf{x} - \mathbf{x}_i\right| \tag{1}$$

which may also be written as

$$H(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \left[(\mathbf{x} - \mathbf{x}_i)^2\right]^{1/2} \tag{2}$$

The coefficients α_i are to be found. The basis functions implied by equations (1) and (2) are not differentiable at the nodes, however. The method was then extended by inclusion of a user-defined input parameter r so that equation (2) is written instead as

$$H(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \left[(\mathbf{x} - \mathbf{x}_i)^2 + r^2\right]^{1/2} \tag{3}$$

This implies that the original basis functions are replaced with equations for a hyperbola.

In order to find the values of the coefficients α_i, one simply applies equation (3) to each of the N points yielding

$$H_i = \sum_{j=1}^{N} \alpha_j \left[(\mathbf{x}_i - \mathbf{x}_j)^2 + r^2 \right]^{1/2} \qquad i = 1, 2, \ldots, N. \tag{4}$$

which can be written in matrix notation as

$$\{H\} = [B]\{\alpha\} \tag{5}$$

where the elements of the $N \times N$ matrix $[B]$ are

$$B_{ij} = \left[(\mathbf{x}_i - \mathbf{x}_j)^2 + r^2 \right]^{1/2} \tag{6}$$

and column matrices $\{H\}$ and $\{\alpha\}$ are given by

$$\{H\} = \left\{ \begin{array}{c} H_1 \\ H_2 \\ \vdots \\ H_N \end{array} \right\} \qquad \{\alpha\} = \left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{array} \right\} \tag{7}$$

Now, given the coordinates of all nodal points and the values of the function H_i at those points, one finds $\{\alpha\}$ as

$$\{\alpha\} = [B]^{-1}\{H\} \tag{8}$$

In 1986 Nelson and Hardy [6] rigorously showed that the MQ method was biharmonic. This proof is important because it shows that the multiquadric representation or approximation does not need to separate data points from source points for optimization. It also leads to the conclusion that the matrix $[B]$ must be invertible. To this end, Franke [7] proposed an inequality (later proved)

$$(-1)^{N-1} \det[B] > 0 \tag{9}$$

This is important since it shows that multiquadrics surface interpolation is always solvable for distinct data.

Franke [7] also introduced a constraint on the coefficients α_j

$$\sum_{j=1}^{N} \alpha_j = 0 \tag{10}$$

to guarantee precision for constant functions.

Implementation

The implementation described here is similar to that used by Kansa [3], which follows Hardy's original scheme closely. First, the undetermined coefficients α_i, $i = 1, 2, \ldots, N$ are found from equation (8). This equation is rewritten assuming that $[B_G]$ is the matrix $[B]$ and $\{H_G\}$ is the column matrix $\{H\}$, and that both are subscripted with the G to emphasize that these matrices are evaluated at the *given* points $\mathbf{x}_{G_i}$. Thus,

$$\{\alpha\} = [B_G]^{-1}\{H_G\} \tag{11}$$

It is desired to find the values of the function H evaluated at the interpolated points $\mathbf{x}_{I_i}$, $i = 1, 2, \ldots, n$. Denoting the column matrix of the values of H at these n points as $\{H_I\}$, we note from equation (3) that

$$\{H_I\} = [B_{IG}]\{\alpha\} = [B_{IG}][B_G]^{-1}\{H_G\} \tag{12}$$

where the elements of $[B_{IG}]$ are

$$B_{IG_{ij}} = \left[\left(\mathbf{x}_{I_i} - \mathbf{x}_{G_j}\right)^2 + r^2\right]^{1/2} \tag{13}$$

Eq. (10) can be implemented either by elimination of one of the α_i's in favor of the others, or it can be imposed via a Lagrange multiplier. The present implementation follows the latter approach, the details of which may be found in Chapter 7 (the TPS formulation) by replacing $[R_G]$ in that formulation by a column matrix $\{U_G\}$ which consists of N ones and $[R_I]$ by a column matrix $\{U_I\}$ which consists of n ones. The result is

$$\{H_I\} = [B_{IG}]\{\alpha\} + \{U_I\}\beta \tag{14}$$

where

$$\{\alpha\} = \frac{\left[[B_G]^{-1} - [B_G]^{-1}\{U_G\}\{U_G\}^T[B_G]^{-1}\right]\{H_G\}}{\{U_G\}^T[B_G]^{-1}\{U_G\}} \tag{15}$$

and

$$\beta = \frac{\{U_G\}^T[B_G]^{-1}\{H_G\}}{\{U_G\}^T[B_G]^{-1}\{U_G\}} \tag{16}$$

Three improvements to this multiquadrics scheme were tested by Kansa [3, 4]. First, he found that increased accuracy could be obtained by allowing the r parameter to vary in such a way as to obtain a well-conditioned matrix. The more distinct the matrix entries, the lower the condition number of the matrix. With this in mind, the r parameter was allowed to vary only monotonically. As stated above, the r parameter controls the shape of the basis function. The increased accuracy did not appear to depend on whether r was monotonically increasing or decreasing. The largest increase in accuracy occurred when r was allowed to vary exponentially. The formula for the variation of r is

$$r_j^2 = r_{\min}^2 \left(\frac{r_{\max}^2}{r_{\min}^2}\right)^{\frac{j-1}{N-1}} \qquad j = 1, 2, \ldots, N \tag{17}$$

where r_j is simply substituted for r in equation (6).

Another improvement was to introduce domain decomposition, which was also implemented here. He found that computational time could be decreased by reducing multiquadrics from a global scheme to a quasi-local scheme using domain decomposition. It was found that using global techniques for multiquadrics caused the coefficient matrix to have large condition numbers. To decrease the condition number, the global domain was broken down into several sub-domains, which were allowed to overlap. Domain decomposition was found to improve computational time and allow for better matrix conditioning. In the present implementation, LU decomposition was used to solve Eq. (15). The domain decomposition was implemented based on the maximum number of points allowed in each coordinate direction. That leads to sub-domains that preserve the original shape and have intersections with the neighboring ones. In the common overlapping regions, all interpolated quantities are determined by using the the mean of the quantities in the pertinent sub-domains.

Kansa finally recommended that for consistent results, regardless the scale of the problems, all data should be mapped onto a unit sub-domain. Therefore, the numerical code maps the data onto a unit line for 1-D problems, onto a unit square for a 2-D problem, and onto a unit cube for a 3-D problem.

Limitations

In the literature [2, 3, 4], it was noted that the multiquadric method provided varying results in many test cases. The method performed poorly for interpolating highly oscillatory functions. However, the same literature points out that multiquadrics, when interpolating the derivatives of a function, performed poorly in regions where the surface is relatively flat, producing results that are somewhat noisy. This can be corrected by use of a hybrid scheme in which multiquadrics is used initially with a corrector method applied to regions of shallow gradients. However, these reported trends were not observed in our research. Finally, the value of the r parameter should be kept within a certain range to insure a stable linear system of equations; see Chapter 11.

References

[1] Hardy, Rolland L., "Multiquadric Equations of Topography and Other Irregular Surfaces.," *Journal of Geophysical Research*, Vol. 76, No. 8, March 1971, pp. 1905 – 1915.

[2] Franke, Richard, "Scattered Data Interpolation: Tests of Some Methods," *Mathematics of Computation*, Vol. 38, No. 157, January 1982, pp. 181 – 200.

[3] Kansa, E. J., "Multiquadrics – A Scattered Data Approximation Scheme with Applications to Computational Fluid-Dynamics – I: Surface Approximations and Partial Derivative Estimates," *Computers and Mathematics with Applications*, Vol. 19, No. 8/9, 1990, pp. 127 – 145.

[4] Kansa, E. J., "Multiquadrics – A Scattered Data Approximation Scheme with Application to Computational Fluid-Dynamics – II: Solutions to Parabolic, Hyperbolic and Elliptic Partial Differential Equations," *Computers and Mathematics with Applications*, Vol. 19, No. 8/9, 1990, pp. 147 – 161.

[5] Hardy, Rolland L., "Theory and Applications of the Multiquadric Biharmonic-Method," *Computers and Mathematics with Applications*, Vol. 19, No. 8/9, 1990, pp. 163 – 208.

[6] Hardy, Rolland L. and Nelson, Stuart A., "A Multiquadric-Biharmonic Representation and Approximation of Disturbing Potential," *Geophysical Research Letters*, Vol. 13, No. 1, January 1986, pp. 18 – 21.

[7] Franke, Richard, "Recent Advances in the Approximation of Surfaces from Scattered Data," In Chui, C.K., Schumaker, L.L., and Utreras, F.I., editors, *Topics in Multivariate Approximation*, pp. 79 – 98. Academic Press, Inc., 1987.

6. NON-UNIFORM B-SPLINE METHOD – MATHEMATICAL FORMULATION

Nomenclature

B	Spline basis function
$C(u)$	Spline curve
$P_0, P_1, \ldots$	Control Points
k	Degree of polynomial (=3 in present work)
m, n	Number of control points
s	Pseudo-arclength along a curve that passes through the knots
$S(u, v)$	Spline surface
u, v	Curvilinear directions describing the surface
u_i, v_i	Knots

Definition

Splines in their most primitive form are used to represent curves in 3-D space, so a tensor product of two splines can be used to represent a surface in 3-D space. Spline functions may either be polynomial or rational in type. Polynomial splines are piecewise polynomial functions of a specified degree. The "basic splines" or B-splines are a subclass of polynomial splines that are linearly independent and span the space of univariate polynomial splines. That is, any polynomial spline function can be represented as a series of B-splines. A rational spline is the ratio of two polynomial splines. Thus, a rational B-spline can be expressed as the ratio of two series of B-splines, the degree of which is the maximum degree of the numerator and denominator. It should be noted that the common term "NURBS" stands for non-uniform rational B-splines and refers to the fact that the knots need not be equally spaced over the parametric interval and that both numerator and denominator have the B-spline representation. In the present work, rational splines are not used – only polynomial cubic splines. (Hence, the name NUBS: non-uniform B-splines).

Current Uses in Aeroelastic Applications

No direct applications of NUBS to aeroelastic methodologies have been found in the literature.

Mathematical Description

Letting $C_0, \ldots, C_{n-1}$ be a set of n points in $\Re^3$, one looks for a spline curve of degree k passing through those points. A spline curve, $C(u)$, with $u \in \Re$, is the mapping of a continuous series of line segments between sequential points, called knots, denoted by $u_0, u_1, \ldots, u_{n+k}$, so that $u_0 \le u \le u_{n+k}$, and where the knots satisfy the relation $u_i \le u_{i+1}$. Thus, for a spline curve of degree k one writes

$$C(u) = \sum_{j=0}^{n-1} B_{j,k}(u) P_j \tag{1}$$

and tries to find the "control points" P_i, where $0 \le i \le n-1$, by solving the linear system

$$C_i = C(u_i) = \sum_{j=0}^{n-1} B_{j,k}(u_i) P_j \tag{2}$$

Although many different types of splines exist, the focus of this work is the "basic" spline (or B-spline) of degree 3.

One defines the i^{th} B-spline function of order $k+1$ (or degree k) as

$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k} - u_i} B_{i,k-1}(u) + \frac{u_{i+k+1} - u}{u_{i+k+1} - u_{i+1}} B_{i+1,k-1}(u) \tag{3}$$

where

$$B_{i,0}(u) = \begin{cases} 1 \text{ if } u_i \le u \le u_{i+1} \text{ and } u_i < u_{i+1} \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

Here, $B_{i,k}(u)$ is a basis function on the interval $u \in [u_0, u_m]$, though the functions are defined along the entire line.

Note that for representation of surfaces, splines are generalized via the tensor product of the representations of two curves to yield a surface $S(u, v)$ given by

$$S(u, v) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} P_{ij} B_{i,k}(u) B_{j,\ell}(v) \tag{5}$$

where $B_{i,k}$ are the B-splines in the u direction (here for each input station direction), $B_{j,\ell}$ are the B-splines in the v direction, P_{ij} are coefficients multiplying these splines in order to fit the data (i.e., control points), m and n correspond to the parameter n of Eq. (1) in the u and v directions, respectively, and k and ℓ are the degrees of the splines for the two directions. This can also be represented as

$$S(u, v) = \sum_{i=1}^{m-1} Q_i(v) B_{ik}(u) \tag{6}$$

where

$$Q_i(v) = \sum_{j=1}^{n-1} P_{ij} B_{j\ell}(v) \tag{7}$$

B-splines have several useful and unique properties [1]:

1. B-spline representations are invariant to linear transformations.

2. A B-spline curve is contained within the convex hull of its neighboring control points. This means that movement of any one control point will result in only a local perturbation.

3. B-spline curves of order k with no multiple interior knots are $k-2$ times differentiable. Note: for Hermite interplation with order k polynomials, we get C^{k-1} continuity, which gives $D^{(k)}$.

4. A B-spline approximation is variation diminishing, meaning that a B-spline approximation will cross the origin no more times than the original function.

5. The B-spline is local in nature with respect to the control net.

A natural spline interpolant is used here, since the derivatives at the nodes and boundaries are unknown, making the use of a complete or Hermite interpolant difficult. The natural spline interpolant is defined as a cubic spline that is continuous on the interval $a \leq x \leq b$ and minimizes the integral

$$I^* = \int_a^b [g''(x)]^2 \, dx \tag{8}$$

The second derivative of the function, $g(x)$, is continuously differentiable and must satisfy the conditions

$$g''(a) = 0 \qquad g''(b) = 0 \tag{9}$$

The first derivatives are subject to

$$g'(a) = A \qquad g'(b) = B \tag{10}$$

where A and B are given constants, which suggests that $g(x)$ is nearly linear at the endpoints a and b [2]. Natural spline end conditions can produce undesirable oscillations when the data vary rapidly [3].

Implementation

The NUBS method is implemented with the aid of a library of NURBS routines called DT_NURBS, developed at the David Taylor Research Center (DTRC) [4]. According to [4], in order to do surface blending, needed in aeroelastic applications, it is recommended that polynomial splines be used because rational splines have a tendency to generate poles and cause numerical problems. Since these routines were originally developed for CAD usage, a main program and surface generating routine were written to implement polynomial splines using the DT_NURBS package.

The primary difference from the CAD applications is the nature of the data that are input. In aeroelasticity applications, the data consist of the grid from either a structural mesh or a CFD

surface grid, along with evaluations of some functions at those grid points that are to be converted to the other grid. While the NURBS library routines for data interpolation require regular grids, aeroelasticity data are not so restricted. Thus, the application of this technique requires some manipulation of the input data.

This manipulation is accomplished via a subroutine based on a similar application by Robert Ames [5] of DTRC. His original FITSURF routine has been modified for use here.

The input data must first be converted to a regular grid. It is assumed that the input data are defined on a semi-structured grid. That is, the grid points at which data are defined can be thought of as being in a two-dimensional array format along the surface. The number of points along each "column" (u-direction) does not have to be identical, and the data in each "row" (v-direction) are not required to lie along constant locations within their respective columns. An example of a semi-structured grid is shown in Fig. 6-1.

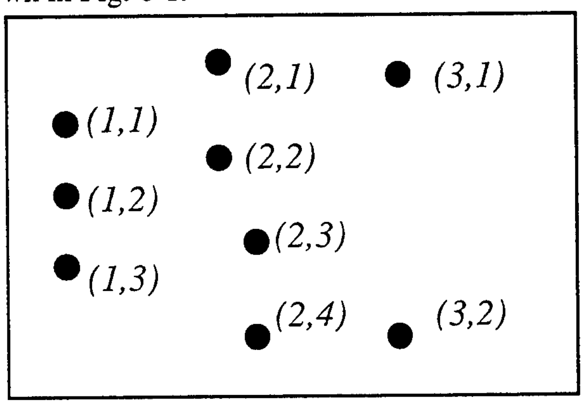


Fig. 6-1: A semi-structured grid

Before spline functions can be created from the non-uniformly spaced grid points and associated data, it is necessary to perform certain transformations to "regularize" the spacing of the grid points. First the maximum number of points in any one column or station is determined and used to generate a regularly spaced set of "knots" from the given grid points. For the sake of accuracy, this must be done such that no surface features are lost. Also, since the DT_NURBS library routines are not capable of handling coincident grid points, the method will stop with an error message if coincident points are discovered.

The next step is to generate a new set of known points on the surface. For each station of data, a multivariate line parameterization routine is called: DTGPAR. In this routine, each line (station) of data is normalized so that it lies within the interval [0, 1] by dividing by the pseudo-arclength parameter s defined as

$$s = \sum_{i=2}^{N} \left[(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2 \right]^{\frac{1}{2}} \tag{11}$$

where N is the total number of points along the station and x_i, y_i, z_i are Cartesian coordinates that define the position of the i^{th} grid point. For large N, s approaches the minimum arclength of a smooth curve drawn through the points. This parameterization is made so that a natural spline can

be fitted to each curve. The natural spline is chosen so that high-order derivatives are zero at the endpoints.

The natural spline is fitted via a subroutine called DTNSI. This subroutine requires that the independent data fall within an interval [0, 1]; hence, the previous call to DTGPAR is made. For the independent variable values and their corresponding dependent variables, a banded matrix of B-spline values is formed at the data points. The linear system of equations that choose the B-spline coefficients to match the input data is factored and then solved.

Through experimentation, it has been determined that a spline of degree 3 is the most robust and accurate for the surfaces under consideration. The linear equations are solved by factoring the banded coefficient matrix using Gaussian elimination, so that coefficients that identify the B-spline are then known.

Finally, the new set of data points that are located at constant parametric locations $u = u_0, u_1, \ldots$ are computed in subroutine DTSEPP, using the univariate spline definition.

The v-direction must also be parameterized. This is accomplished by applying the first point for each knot along the v-axis and calling subroutine DTGPAR (see previous description).

Finally, a new surface that incorporates all of the newly splined data needs to be defined. This is accomplished by a call to subroutine DTCRBL. DTCRBL constructs a tensor product spline surface from the curves previously generated at constant parameter lines based on Eq. (5). This permits the same use of DTNSI as before by parametrizing the surface as described in Eq. (5).

Limitations

- There must be at least 4 curves and at least 4 points.
- Contiguous data points can be coincident in two of the three directions, but not all three (i.e., no surface knuckles or chimes).
- Degenerate data without C^0 continuity will be "smoothed over" and C^0 continuity is enforced.

The following limitation on the search routine should also be noted. At this point, the new dependent data variables can be evaluated at the new grid points. Because these routines require that the data points lie in the interval [0, 1], a multivariate interpolation is used to convert the x, y, z coordinates of the unknown data point to this interval with respect to the original data and its parameterization. This last caveat is extremely important. If the two grids are not directly coincident, independent parameterization of the new data set will lead to a skewed and inaccurate representation.

References

[1] Piegl, L. and Tiller, W., "Curve and Surface Constructions Using Rational B-splines," *Computer Aided Design*, Vol. 19, No. 9, November 1987, pp. 485 – 495.

[2] Kreyszig, E., *Advanced Engineering Mathematics*, John Wiley & Sons, New York, 1988.

[3] Foley, T. A., "Weighted Bicubic Spline Interpolation to Rapidly Varying Data," *ACM Transactions on Graphics*, Vol. 6, No. 1, January 1987, pp. 1 – 18.

[4] Anon., *DT_NURBS Spline Geometry Subprogram Library User's Manual*, Boeing Computer Services, version 2.3 Edition, February 1995, pp. 3-1 – 4-5.

[5] Ames, R., "FITSURF 6.0," Private correspondence, 1995.

7. THIN-PLATE SPLINE METHOD – MATHEMATICAL FORMULATION

Nomenclature

a	A set of known points in A
A	A surface in $\Re^\ell$
α_i	Undetermined coefficients
$\{\alpha\}$	Column matrix of undetermined coefficients
$\beta_0,\beta_x,\beta_y,\beta_z$	Undetermined coefficients
$\{\beta\}$	Column matrix of undetermined coefficients
$[B]$, $[B_G]$	Matrices constructed from basis functions evaluated at given points
$[B_{IG}]$	Coupling matrix constructed from basis functions evaluated at given and interpolated points
D	Differential operator
$H(\mathbf{x})$	Function to be interpolated
$\{H\}$, $\{H_G\}$	Column matrices of given values of the function to be interpolated
$\{H_I\}$	Column matrix of interpolated values of the function
$K_{2m+2s-\ell}$	A generic function used to define the spline functions
ℓ	Order of the space dimensionality
m	Order of the partial derivatives in the energy function
n	Total number of points to be interpolated
N	Total number of given points
$[R]$, $[R_G]$	Matrices containing coordinates of the given points
$[R_I]$	Matrices containing coordinates of the points to be interpolated
s	Parameter needed to define the type and/or order of K
t	Generic independent variable
$\mathbf{x}$	Position vector to an arbitrary point in Cartesian space
$\mathbf{x}_i$, $\mathbf{x}_{G_i}$	Position vector to the ith given points in Cartesian space
$\mathbf{x}_{I_i}$	Position vector to the ith point to be interpolated in Cartesian space

Definition

The method of thin-plate splines (TPS), also called surface splines, provides a means to characterize an irregular surface by using functions that minimize an energy functional [4]. This methodology is very similar to the multiquadric-biharmonic method developed by Hardy [1]. The primary difference in these two methods is the basis functions used. Here, the problem is approached from an engineering or physical representation of the surface. That is, for a one-dimensional (1-D) problem,

elementary cubic splines can be interpreted as equilibrium positions of a beam undergoing bending deformation [2]. For a 2-D problem (such as a surface), these splines can be determined from the minimization of the bending energy (thus defining the equilibrium position) of a thin plate. Since these types of splines are invariant with rotation and translation, they are very powerful tools for the interpolation of moving or flexible surfaces [3]. It should be noted that the infinite-plate splines method is a special case of the TPS method: for 2-D they are exactly the same mathematically.

Current Uses in Aeroelastic Applications

Specific aeroelastic applications are unknown for implementation as a method more general than its 2-D form, which is essentially the same as the IPS method discussed in Chapter 4.

Mathematical Description

Elementary cubic splines characterize the static equilibrium configuration of a slender beam [2]. Thin-plate splines minimize similar functionals generalized to 2-D of a thin plate. There are also generalized versions applicable to higher-dimensional problems. The methodology defined here was verified mathematically by Duchon [3, 4]. The bending energy for a thin plate can be written as

$$\Phi = \int_{\Omega} |D^2 v|^2 d\Omega \tag{1}$$

where Ω is the domain of the plate, v is the displacement of the plate, and D is a partial differential operator. The splines that minimize this function are difficult to compute. Duchon [4] extended the functional to multi-dimensional domains wherein a generalized version of Eq. (1) now becomes

$$\Phi = \int_{\Re^{\ell}} |D^m v|^2 d\Re^{\ell} \tag{2}$$

When subject to the restriction that $m > \ell/2$ where point values are used (as is the case here), this functional is very easy to solve.

Duchon's extension of Eq. (1) to Eq. (2) also has the advantage that the functional in Eq. (2) is invariant with respect to translations and rotations. Moreover, if a similarity transformation $(t \rightarrow \alpha t)$ is applied to the surface v, the interpolation is multiplied by some power of $|\alpha|$. Therefore the interpolation can be applied to a nondimensionalized set of data and will obtain the same result as applying the interpolation to the original data. It is very important that only one solution exists to the minimization of the functional in Eq. (2). This uniqueness was proven by Duchon [3] for solutions which are of the form

$$\sigma(t) = \sum_{a \in A} \alpha_a K_{2m+2s-\ell}(t-a) + p(t) \tag{3}$$

where a represents a set of points that lie within a finite surface $A \subset \Re^{\ell}$ and $p \in P_{m-1}$ where P_{m-1} is a unisolvent subset of A. The dependence of the generic function $K_{2m+2s-\ell}$ on the parameters m

and ℓ, as well as another s, is reflected in the subscript of K to describe its type and order. It must also be true that

$$\sum_{a \in A} \alpha_a q(a) = 0 \tag{4}$$

where the function q and its results are also a part of the subset P_{m-1} ($\forall q \in P_{m-1}$) for the set of prescribed points in A.

Duchon [3] has shown that in order to describe a thin-plate function, the parameters should be set to $m = 2$, $s = 0$, and $\ell = 2$. This yields a specific form for the generic function so that

$$\sigma(t) = \beta_0 + \beta_1 t + \sum_{a \in A} \alpha_a |t - a|^2 \log |t - a| \tag{5}$$

with $\sum_{a \in A} \alpha_a = 0$ and $\sum_{a \in A} \alpha_a a = 0$ in accordance with Eq. (4). So that the function can be considered to be continuous, the function $|t - a|^2 \log |t - a|$ is extended to include $a = 0$ in accordance with its limiting value of 0. The coefficients of the low-order polynomials can be found as indicated by [3]. These additions to the primal formulation of the TPS method lead to a set of functions that minimize the norms of the second derivatives of the interpolant. The surface defined by A is subject to the limitation that it must not be contained totally in a line or a point.

The thin-plate function of Eq. (5) can be used to construct a function $H(\mathbf{x})$, given its values at a set of N discrete "nodal" values $H_i = H(\mathbf{x}_i)$ at "nodes" $\mathbf{x} = \mathbf{x}_i$ for $i = 1, 2, \ldots, N$. Here the vector $\mathbf{x} = x\check{\mathbf{i}} + y\check{\mathbf{j}} + z\check{\mathbf{k}}$ with unit vectors $\check{\mathbf{i}}$, $\check{\mathbf{j}}$, and $\check{\mathbf{k}}$ along the Cartesian coordinate directions x, y, and z, respectively. This describes the domain $\Re^3$.

The interpolation function can be written as

$$H(\mathbf{x}) = \beta_0 + \beta_x x + \beta_y y + \beta_z z + \sum_{i=1}^{N} \alpha_i |\mathbf{x} - \mathbf{x}_i|^2 \log |\mathbf{x} - \mathbf{x}_i| \tag{6}$$

where the scalar term $|\mathbf{x} - \mathbf{x}_i|^2$ is defined as $(\mathbf{x} - \mathbf{x}_i) \cdot (\mathbf{x} - \mathbf{x}_i)$. (It is noted that, when specialized to 2-D, the identical mathematical equation is found in Eq. (10) of Chapter 4.)

The coefficients α_i, for $i = 1, 2, \ldots, N$, and β_0, β_x, β_y, and β_z are to be determined by solution of the minimization problem. The result is that these coefficients can be found by applying Eq. (6) to each of the N nodes yielding

$$H_i = \beta_0 + \beta_x x_i + \beta_y y_i + \beta_z z_i + \sum_{j=1}^{N} \alpha_j |\mathbf{x}_i - \mathbf{x}_j|^2 \log |\mathbf{x}_i - \mathbf{x}_j| \tag{7}$$

subject to the side conditions

$$\sum_{i=1}^{N} \alpha_i = \sum_{i=1}^{N} \alpha_i x_i = \sum_{i=1}^{N} \alpha_i y_i = \sum_{i=1}^{N} \alpha_i z_i = 0 \tag{8}$$

In order to solve for the unknown coefficients, one can introduce a matrix notation such that Eqs. (7) and (8) become

$$\{H\} = [B]\{\alpha\} + [R]\{\beta\} \qquad\qquad [R]^T\{\alpha\} = 0 \tag{9}$$

where the elements of the $N \times N$ matrix $[B]$ are

$$B_{ij} = |\mathbf{x}_i - \mathbf{x}_j|^2 \log |\mathbf{x}_i - \mathbf{x}_j| \tag{10}$$

the matrix $[R]$ is

$$[R] = \begin{bmatrix} \begin{Bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{Bmatrix} & \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{Bmatrix} & \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{Bmatrix} & \begin{Bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{Bmatrix} \end{bmatrix} \tag{11}$$

and column matrices $\{H\}$, $\{\alpha\}$, and $\{\beta\}$ are given by

$$\{H\} = \begin{Bmatrix} H_1 \\ H_2 \\ \vdots \\ H_N \end{Bmatrix} \qquad \{\alpha\} = \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{Bmatrix} \qquad \{\beta\} = \begin{Bmatrix} \beta_0 \\ \beta_x \\ \beta_y \\ \beta_z \end{Bmatrix} \tag{12}$$

Now, given the coordinates of all nodal points and the values of the function H_i at those points, one finds $\{\alpha\}$ as

$$\{\alpha\} = \left[[B]^{-1} - [B]^{-1}[R] \left[[R]^T [B]^{-1} [R] \right]^{-1} [R]^T [B]^{-1} \right] \{H\} \tag{13}$$

Implementation

The implementation into a computer code is straightforward. First, the undetermined coefficients α_i, β_x, β_y, β_z, and β_0 are found from equation (13). This equation is rewritten assuming that $[B_G]$ is the matrix $[B]$, that $[R_G]$ is the matrix $[R]$ and that $\{H_G\}$ is the column matrix $\{H\}$; they are subscripted with the G to emphasize that these matrices are evaluated at the *given* points $\mathbf{x}_{G_i}$. Thus,

$$\{\alpha\} = \left[[B_G]^{-1} - [B_G]^{-1}[R_G] \left[[R_G]^T [B_G]^{-1} [R_G] \right]^{-1} [R_G]^T [B_G]^{-1} \right] \{H_G\} \tag{14}$$

It is desired to find the values of the function H evaluated at the interpolated points $\mathbf{x}_{I_i}$, $i = 1, 2, \ldots, n$. Denoting the column matrix of the values of H at these n points as $\{H_I\}$, we note from the first of Eqs. (9) that

$$\{H_I\} = [B_{IG}]\{\alpha\} + [R_I]\{\beta\} \tag{15}$$

where the elements of $[B_{IG}]$ are

$$B_{IG_{ij}} = |\mathbf{x}_{I_i} - \mathbf{x}_{G_j}|^2 \log |\mathbf{x}_{I_i} - \mathbf{x}_{G_j}| \tag{16}$$

$[R_I]$ is given by

$$[R_I] = \begin{bmatrix} \begin{Bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{Bmatrix} & \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} & \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{Bmatrix} & \begin{Bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{Bmatrix} \end{bmatrix} \tag{17}$$

and $\{\beta\}$ is

$$\{\beta\} = \left[[R_G]^T[B_G]^{-1}[R_G]\right]^{-1}[R_G]^T[B_G]^{-1}\{H_G\} \tag{18}$$

Limitations

When specialized to 2-D and limited to a global implementation, the TPS method has the same limitations as the method of infinite plate splines (IPS). However, when generalized to 3-D and when implemented as a local method (i.e., with sub-domaining), it outperforms the IPS method. The dimension of the interpolant has to coincide with the dimension of the (sub) domain being interpolated. If the (sub) domain collaps to a lower order (all points along the same line in a original 2-D interpolation, or all points on the same plane in a 3-D interpolation), the side condition matrix $[R]$ will have proportional rows, making the overall interpolation scheme unstable. The points within a (sub) domain has to first be preprocessed, eventually rotated, and only then the order of the interpolant is defined.

References

[1] Hardy, Rolland L., "Multiquadric Equations of Topography and Other Irregular Surfaces.," *Journal of Geophysical Research*, Vol. 76, No. 8, March 1971, pp. 1905 – 1915.

[2] Atteia, M., "Fonctions 'Spline' et Noyaux Reproduisants d'Aronszajn-Bergman," *Revue Française d'Informatique et de Recherche Opérationelle*, Vol. R-3, 1970, pp. 31 – 43.

[3] Duchon, Jean, "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," In Schempp, W. and Zeller, K., editors, *Constructive Theory of Functions of Several Variables, Oberwolfach 1976*, pp. 85 – 100. Springer-Verlag, Berlin, 1977.

[4] Duchon, Jean, "Fonctions-Spline à Énergie Invariante par Rotation," Technical Report R. R. No. 27, Université de Grenoble, January 1976.

[5] Franke, Richard, "Scattered Data Interpolation: Tests of Some Methods," *Mathematics of Computation*, Vol. 38, No. 157, January 1982, pp. 181 – 200.

8. FINITE-PLATE SPLINE METHOD – MATHEMATICAL FORMULATION

Nomenclature

n	Number of structural grid points
m	Number of aerodynamic grid points
N	Number of grid points in virtual surface
x, y, z	Coordinates of a generic point (Cartesian)
$w(x, y)$	Displacement along the z axis
$\theta(x, y)$	Rotation about the x axis
$\phi(x, y)$	Rotation about the y axis
$\{r\}$	Column matrix of displacement and rotations
$[\Omega]$	Matrix of shape functions and their derivatives
$\{q^{(e)}\}$	Column matrix of local element displacements and rotations
$\lfloor\omega\rfloor$	Row matrix of shape functions
$\{q\}$	Column matrix of global displacements and rotations at virtual points
$[C]$	Connectivity matrix
$\{q_s\}$	Column matrix of displacements and rotations at structural points
$[\psi_s]$	Matrix of shape functions evaluated at the structural nodes
$\{q_a\}$	Column matrix of displacements at aerodynamic points
$[\psi_a]$	Matrix of shape functions evaluated at the aerodynamic grids
$[K]$	Free-free stiffness of the plate
$[\alpha]$	Diagonal matrix of weighting constants
$[T]$	Mapping matrix relating structural and aerodynamic grid points
$\{F_a\}$	Column matrix of aerodynamic loads at the aerodynamic grid points
$\{F_s\}$	Column matrix of loads at the structural grid points
$\{\delta q_s\}$	Column matrix of virtual displacements at structural grid points
$\{\delta q_a\}$	Column matrix of virtual displacements at aerodynamic grid points
ξ,η	Nondimensional coordinates
a,b	Dimensions of the quadrilateral element
$c_0, c_1, \ldots, c_{11}$	Undetermined coefficients
$\overline{(\,.\,)}$	Non-dimensional form of quantity (.)
ℓ	Reference length
$[N]$	Matrix of shape functions for a 4-node quadrilateral element
$[Y]$	Matrix of constants

Definition

The original finite-plate spline (FPS) method of Appa [1] employs uniform plate elements to represent a given planform by a number of quadrilateral or triangular bending elements. A virtual surface is defined and constrained to pass through both structural and aerodynamic grid points. These constraints are imposed at the element level, and a proper choice of shape functions is required. The shape functions define a virtual surface that relates displacements at the structural and aerodynamic grid points. The node points of the virtual surface generated in this method to interface the two boundaries does not have to coincide with the number of grid points of either mesh. Usually, however, the grid points of the virtual surface are a subset of the structural grid points.

The FPS method has the advantage of accomodating changes in fluid and structural models easily. In addition, this approach conserves the work done by the aerodynamic forces when obtaining the global nodal force vector. Finally, as a finite-element-based method, it is sufficiently versatile to accurately model realistic body geometries.

Current Uses in Aeroelastic Applications

The original formulation of Appa [1] seems to be still in use at Northrop Corporation. A 3-D extension of it has been incorporated in ENSAERO (NASA Ames) as briefly described in Refs. [3] and [4], but no details can be found in the literature.

Mathematical Description

In this section, a linear mapping matrix is derived using the structural finite element method, based on the minimum energy principle. The first result is a direct compilation of the work described in [1] that only deals with an out-of-plane bending of the virtual plate. Later, a generalization of that for 3-D displacement is presented.

Consider a wing representation such that the wing lies in the x-y plane, and its surface is divided in subdomains where the finite element for interpolation is applied. We assume that there are N points in this discretization, and those points are not necessarily coincident either with the m aerodynamic points or with the n structural displacement points.

Even though not restricted to this topology, consider a four-noded quadrilateral element. The displacement $w(x, y)$, along the z direction, and the rotations $\theta(x, y)$, about the x axis, and $\phi(x, y)$, about the y axis, at any point (x, y) within the element, are given by

$$\{r\} = [\Omega]\left\{q^{(e)}\right\} \tag{1}$$

where

$$\{r\} = \left\{ \begin{array}{c} w \\ \theta \\ \phi \end{array} \right\} \qquad\qquad [\Omega] = \left\{ \begin{array}{c} \lfloor \omega \rfloor \\ \lfloor \omega_{,y} \rfloor \\ \lfloor \omega_{,x} \rfloor \end{array} \right\} \tag{2}$$

$$\left\{q^{(e)}\right\} = \left\lfloor\, w_1 \;\; \theta_1 \;\; \phi_1 \;\; \cdots \;\; w_4 \;\; \theta_4 \;\; \phi_4 \,\right\rfloor^T$$

and where the angles satisfy the relation

$$\theta = \frac{\partial w}{\partial y} \qquad\qquad \phi = \frac{\partial w}{\partial x} \tag{3}$$

In the above, $\lfloor \omega \rfloor$ is a (1×12) row matrix of shape functions used to interpolate the displacement field within the element in terms of its nodal degrees of freedom $\left\{q^{(e)}\right\}$. As suggested by Appa [1], the C^1 shape functions given by Argyris [5] are a natural option (see "Implementation" below).

The local displacement column matrix $\left\{q^{(e)}\right\}$ can always be related to the displacement column matrix of the virtual mesh $\{q\}$ (an $N \times 1$ column matrix) by means of a certain connectivity matrix $[C]$ (see [6]). Therefore, for the i^{th} element

$$\left\{q^{(e)}\right\}_i = [C]_i\{q\} \tag{4}$$

With Eqs. (1) and (4), each of the n structural points can be evaluated in terms of the displacement column matrix of the virtual mesh $\{q\}$ and assembled in the final column matrix $\{q_s\}$. If $[\psi_s]$ represents the assembled matrix for the local shape functions evaluated at the structural nodes, and simultaneously related to the virtual surface, *i.e.*,

$$[\psi_s] = \begin{bmatrix} [\Omega]_1[C]_1 \\ [\Omega]_2[C]_2 \\ \vdots \\ [\Omega]_n[C]_n \end{bmatrix} \tag{5}$$

then the structural displacement column matrix $\{q_s\}$ is written as

$$\{q_s\} = [\psi_s]\{q\} \tag{6}$$

Similarly, the aerodynamic displacement column matrix $\{q_a\}$ can be written as

$$\{q_a\} = [\psi_a]\{q\} \tag{7}$$

Since the virtual surface is required to pass through the given set of structural points $\{q_s\}$, the penalty method [6] is employed. The equilibrium state of the virtual structure is

$$[K]\{q\} + [\alpha][\psi_s]^T\left([\psi_s]\{q\} - \{q_s\}\right) = 0 \tag{8}$$

where $[K]$ is the free-free stiffness of the plate and $[\alpha]$ is a diagonal matrix of weighting constants that can be used to scale the elements of $[K]$ and $[\psi_s]^T[\psi_s]$.

Eq. (8) can be put in the form

$$\left([\alpha]^{-1}[K] + [\psi_s]^T[\psi_s]\right)\{q\} = [\psi_s]^T \{q_s\} \tag{9}$$

and, therefore, symbolically solved for $\{q\}$ as

$$\{q\} = \left([\alpha]^{-1}[K] + [\psi_s]^T[\psi_s]\right)^{-1} [\psi_s]^T \{q_s\} \tag{10}$$

Note here that even though the matrix $[K]$ is originally singular, the addition of the constraint matrix makes the total matrix non-singular.

Finally, using Eq. (10) in Eq. (7), one obtains

$$\{q_a\} = [\psi_a]\left([\alpha]^{-1}[K] + [\psi_s]^T[\psi_s]\right)^{-1} [\psi_s]^T \{q_s\} \tag{11}$$

Therefore, the $(3m \times 3n)$ mapping matrix relating the structural and aerodynamic displacements is given by

$$[T] = [\psi_a]\left([\alpha]^{-1}[K] + [\psi_s]^T[\psi_s]\right)^{-1} [\psi_s]^T \tag{12}$$

Now, consider the distribution of the aerodynamic load $\{F_a\}$ on the n structural grid points. From the principle of virtual work, the following relation holds [7]

$$\{\delta q_s\}^T \{F_s\} = \{\delta q_a\}^T \{F_a\} \tag{13}$$

where $\{F_s\}$ is the load column matrix corresponding to the structural grid points.

Since the virtual displacements have to be compatible, substituting the variation of Eq. (11) into Eq. (13), one obtains

$$\{\delta q_s\}^T \{F_s\} = \{\delta q_s\}^T [T]^T \{F_a\} \tag{14}$$

and therefore,

$$\{F_s\} = [T]^T \{F_a\} \tag{15}$$

where $[T]^T$ is the load transformation matrix.

Implementation

Several types of bending elements could be used in this formulation described in the previous section. As discussed in [1], a C^0-type element that employs independent interpolation functions for displacement and rotations did not yield satisfactory results. It is recommended to use a C^1-type element. The one used in [1] that provided good results is based on the natural modes proposed by Argyris [5].

Consider a quadrilateral element and an associated local Cartesian system. The nondimensional coordinates are defined as $\xi = \frac{2x}{a}$ and $\eta = \frac{2x}{b}$, where a and b are the dimensions of the quadrilateral

element (Fig. 8-2). The interpolation polynomial is for a cubic serendipity element [6], and is given by

$$\overline{w} = c_0 + c_1\xi + c_2\eta + c_3\xi\eta + c_4\xi^2 + c_5\eta^2 + + c_6\xi^2\eta + c_7\xi\eta^2 + c_8\xi^3 + c_9\eta^3 + c_{10}\xi^3\eta + c_{11}\xi\eta^3 \quad (16)$$

where $\overline{w} = \frac{w}{\ell}$, with ℓ being a reference length. Similarly, consider the non-dimensional form of the rotation

$$\overline{\theta} = \frac{\partial \overline{w}}{\partial \eta} \qquad \overline{\phi} = \frac{\partial \overline{w}}{\partial \xi} \quad (17)$$

The non-dimensional measures can be related to the corresponding dimensional ones (Eq. 2) by

$$\{\overline{r}\} = \left\{ \begin{array}{c} \overline{w} \\ \overline{\theta} \\ \overline{\phi} \end{array} \right\} = \begin{bmatrix} \frac{1}{\ell} & 0 & 0 \\ 0 & \frac{b}{2\ell} & 0 \\ 0 & 0 & \frac{a}{2\ell} \end{bmatrix} \{r\} \quad (18)$$

With the relations defined by Eq. (17), one can write

$$\{\overline{r}\} = [N]\{c\} \quad (19)$$

where

$$[N] = \begin{bmatrix} 1 & \xi & \eta & \xi\eta & \xi^2 & \eta^2 & \xi^2\eta & \xi\eta^2 & \xi^3 & \eta^3 & \xi^3\eta & \xi\eta^3 \\ 0 & 0 & 2 & 2\xi & 0 & 4\eta & 2\xi^2 & 4\xi\eta & 0 & 6\eta^2 & 2\xi^3 & 6\xi\eta^2 \\ 0 & -2 & 0 & -2\eta & -4\xi & 0 & -4\xi\eta & -2\eta^2 & -6\xi^2 & 0 & -6\xi^2\eta & -2\eta^3 \end{bmatrix} \quad (20)$$

and

$$\{c\} = \left\{ \begin{array}{ccccc} c_0 & c_1 & c_2 & \cdots & c_{11} \end{array} \right\}^T \quad (21)$$

In order to have the shape functions in a more suitable form, consider the solution of the constants defined in Eq. (21) as function of the nondimensional nodal values $\{\overline{q^{(e)}}\}$. Hence,

$$\{c\} = [Y]\{\overline{q^{(e)}}\} \quad (22)$$

where

$$[Y] = \begin{bmatrix} \frac{1}{4} & \frac{1}{16} & -\frac{1}{16} & \frac{1}{4} & \frac{1}{16} & \frac{1}{16} & \frac{1}{4} & -\frac{1}{16} & \frac{1}{16} & \frac{1}{4} & -\frac{1}{16} & -\frac{1}{16} \\ -\frac{3}{8} & -\frac{1}{16} & \frac{1}{16} & \frac{3}{8} & \frac{1}{16} & \frac{1}{16} & \frac{3}{8} & -\frac{1}{16} & \frac{1}{16} & -\frac{3}{8} & \frac{1}{16} & \frac{1}{16} \\ -\frac{3}{8} & -\frac{1}{16} & \frac{1}{16} & -\frac{3}{8} & -\frac{1}{16} & -\frac{1}{16} & \frac{3}{8} & -\frac{1}{16} & \frac{1}{16} & \frac{3}{8} & -\frac{1}{16} & -\frac{1}{16} \\ \frac{1}{2} & \frac{1}{16} & -\frac{1}{16} & -\frac{1}{2} & -\frac{1}{16} & -\frac{1}{16} & \frac{1}{2} & -\frac{1}{16} & \frac{1}{16} & -\frac{1}{2} & \frac{1}{16} & \frac{1}{16} \\ 0 & 0 & \frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 & 0 & \frac{1}{16} \\ 0 & -\frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 & 0 & \frac{1}{16} & 0 & 0 & \frac{1}{16} & 0 \\ 0 & 0 & -\frac{1}{16} & 0 & 0 & \frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 & 0 & \frac{1}{16} \\ 0 & \frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 & 0 & \frac{1}{16} & 0 & 0 & -\frac{1}{16} & 0 \\ \frac{1}{8} & 0 & -\frac{1}{16} & -\frac{1}{8} & 0 & -\frac{1}{16} & -\frac{1}{8} & 0 & -\frac{1}{16} & \frac{1}{8} & 0 & -\frac{1}{16} \\ \frac{1}{8} & \frac{1}{16} & 0 & \frac{1}{8} & \frac{1}{16} & 0 & -\frac{1}{8} & \frac{1}{16} & 0 & -\frac{1}{8} & \frac{1}{16} & 0 \\ -\frac{1}{8} & 0 & \frac{1}{16} & \frac{1}{8} & 0 & \frac{1}{16} & -\frac{1}{8} & 0 & -\frac{1}{16} & \frac{1}{8} & 0 & -\frac{1}{16} \\ -\frac{1}{8} & -\frac{1}{16} & 0 & \frac{1}{8} & \frac{1}{16} & 0 & -\frac{1}{8} & \frac{1}{16} & 0 & \frac{1}{8} & -\frac{1}{16} & 0 \end{bmatrix} \quad (23)$$

Finally, combining Eqs. (19) and (22), one can compute $[\Omega]$ from Eq. (2) as

$$[\Omega] = [N][Y] \tag{24}$$

Note: From Eq. (18), it directly follows that $\{\overline{r}\} = [\Omega]\{\overline{q^{(e)}}\}$.

Limitations

As the method is quite new, few specific limitations are known other than its requiring large amounts of memory and CPU time.

References

[1] Appa, Kari, "Finite-Surface Spline," *Journal of Aircraft*, Vol. 26, No. 5, May 1989, pp. 495 – 496.

[2] Harder, R. L. and Desmarais, R. N., "Interpolation Using Surface Splines," *Journal of Aircraft*, Vol. 9, No. 2, 1972, pp. 189 – 191.

[3] Guruswamy, G. P. and Byun, C., "Fluid-Structural Interactions Using Navier-Stokes Flow Equations Coupled with Shell Finite Element Structures," In *Proceedings of the 24th Fluid Dynamics Conference*. AIAA, July 6 – 9 1993, AIAA Paper 93-3087.

[4] Guruswamy, G. P. and Byun, C., "Direct Coupling of Euler Flow Equations with Plate Finite Element Structures," *AIAA Journal*, Vol. 33, No. 2, February 1995, pp. 375 – 377.

[5] Argyris, J. H., "Matrix Displacement Analysis of Plates and Shells - Prolegomena to a General Theory, Part I," *Ingenieur-Archiv*, Vol. 35, No. 2, 1966, pp. 102 – 142.

[6] Bathe, K. J., *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1982.

[7] Appa, Kari, Yankulich, Micheal, and Cowan, David L., "The Determination of Load and Slope Transformation Matrices for Aeroelastic Analyses," *Journal of Aircraft*, Vol. 22, No. 8, August 1985, pp. 734 – 736.

[8] Cook, R. D., Malkus, D. S., and Plesha, M. E., *Concepts and Applications of Finite Element Analysis*, John Wiley and Sons, New York, 3rd Edition, 1989.

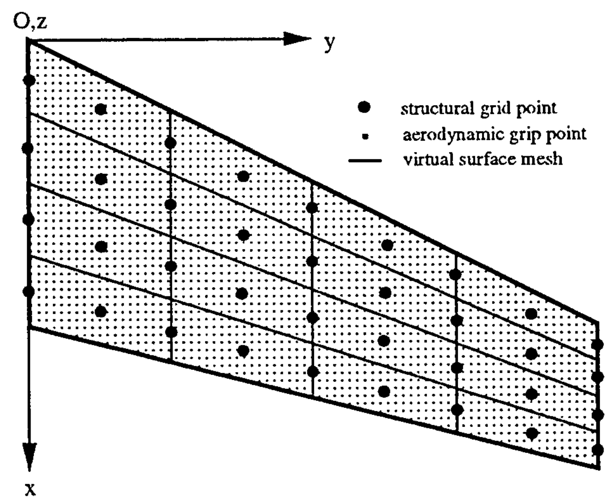


Fig. 8-1: Representation of a wing with two distinct sets of grid points and the virtual mesh

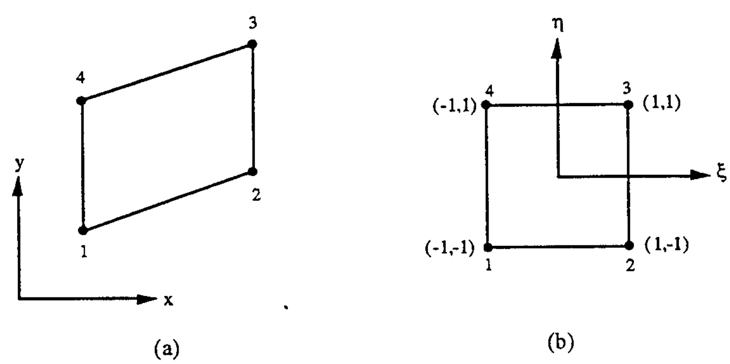


Fig. 8-2: (a) Quadrilateral element in $x - y$ plane; (b) Element in $\xi - \eta$ plane

9. INVERSE ISOPARAMETRIC MAPPING METHOD – MATHEMATICAL FORMULATION

Nomenclature

A, B, C	Coefficients of quadratic equation
a	Slope of a straight line
$a_{x_1}, \ldots, a_{x_4}$	Intermediate constants
$a_{y_1}, \ldots, a_{y_4}$	Intermediate constants
c_P	Constant associated with point P
n	Number of iterations
N	Shape function
M	Interior point; aerodynamic point
P	Corner node
Q	Side point
u	Displacement field; any field
u_i^e	Field value at nodal point i
x, y	Cartesian coordinates
$\mathbf{x}$	Pair (x, y)
x_i^e, y_i^e	Cartesian coordinates of nodal point i
$\mathbf{x}_i^e$	Pair (x_i^e, y_i^e)
δ_n	Tolerance on the error after n iterations
ξ, η	Local coordinates
$\boldsymbol{\xi}$	Pair (ξ, η)

Definition

The Inverse Isoparametric Mapping (IIM) method is based on finite element analysis where an isoparametric element uses the same shape functions (N_i) to interpolate both the coordinate and the displacement vectors [1]. This interpolation is a one-to-one mapping, termed isoparametric, from a local ($\boldsymbol{\xi}$) to a global Cartesian ($\mathbf{x}$) or a displacement (u) plane. In mathematical form

$$x = \sum_i N_i(\boldsymbol{\xi}) x_i^e \qquad y = \sum_i N_i(\boldsymbol{\xi}) y_i^e \qquad u = \sum_i N_i(\boldsymbol{\xi}) u_i^e \tag{1}$$

where x_i^e, y_i^e are Cartesian coordinates of the structural nodal point i, and $\mathbf{x} = (x, y)$, $\boldsymbol{\xi} = (\xi, \eta)$. On the other hand, if one has a given point in the global domain and wants the local coordinates corresponding to it, the inverse of this mapping involves a system of nonlinear equations, even for

the linear strain element. The system of nonlinear equations can be solved numerically using an iterative approach.

Current Uses in Aeroelastic Applications

As discussed in [2], IIM is often used in dynamic finite element analyses (*e.g.*, remeshing), and highly nonlinear dynamic analyses (*e.g.*, nodal contouring). Pidaparti [3] presents the method in the context of aeroelastic analysis, even though no significant result is presented. At NASA Ames, Guruswamy and co-workers implemented this method in one of the ENSAERO versions [4].

Mathematical Description

As mentioned above, the idea of IIM is to find local coordinates from the global coordinates of a certain point M. In general, this cannot be obtained analytically. Instead of solving a system of nonlinear equations, a direct iterative technique of order n^2 is often employed. An improved version of it was presented by Murti and Valliappan [2], in which the iteration is reduced to order n. The iteration is improved by bisecting a defined line that passes through a point M of coordinate $\mathbf{x}_m$ and a node of known local coordinate ξ_p such as a corner node.

Following [2], let us consider the straight line in the global frame, PQ, that passes through the point M. The point Q is defined just by the intersection of the straight line with the side of the element. The mapping of this line into the transformed plane has an associated image $P'Q'$, which generally is a parabolic curve. By searching along the curve $P'Q'$, the local coordinates of the point M can be found.

Consider

$$PQ \equiv y = ax + c_p \tag{2}$$

where

$$a = \frac{(y_m - y_p)}{(x_m - x_p)} \qquad c_p = y_p - ax_p \tag{3}$$

Using Eq. (1) in Eq. (2), one obtains

$$\sum_i N_i y_i = a \sum_i N_i x_i + c_p \tag{4}$$

where i varies from 1 to the total number of nodes in the element. This can be simplified as

$$\sum_i N_i c_i - c_p = 0 \tag{5}$$

with $c_i = y_i - ax_i$.

Note that, since c_i and c_p are constants for a given M and nodal coordinates (x_i, y_i), Eq. (5) can be rearranged as

$$A\xi^2 + B\xi + C = 0 \tag{6}$$

such that $\xi, \eta \in [-1, +1]$, and $A, B, C = f(c_i, c_p, \eta)$. Explicit expressions for these coefficients are given in [2] for elements with the number of nodes varying between 4 and 9.

One can now carry out the iteration by selecting η_0 and evaluating the coefficients A, B, and C, so that, if $A \neq 0$, Eq. (6) has solutions

$$\xi_0 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{7}$$

Between the two solutions, only one will be in the interval $[-1, +1]$, since there is a one-to-one correspondence between the global and local frames. After n iterations and tolerance δ_n, one gets

$$\mathbf{x}_m = \sum_i N_i(\xi_m^n)\mathbf{x}_i^e + \delta_n \tag{8}$$

After convergence has been achieved for the local coordinates (ξ_m, η_m), then one can directly use Eq. (1) to do the interpolation.

There are some variants to the solution presented above. For example, when $P'Q'$ is not defined in the entire range of $\eta \in [-1, +1]$, a change of corner nodes should be imposed. Another anomaly happens when the abscissa of M is equal to the abscissa of the chosen corner node. This will make $A = 0$, and redefinition of PQ is also required.

Implementation

The implementation used in our studies is the one done by Fithen [5]. It uses a bilinear (4-node) element and no information of the original structural mesh is taken into account to determine the cell that contains a given aerodynamic point. Unfortunately, there is no written documentation of the code. We were unable to establish a direct communication with the author, and the only source of information was the program's FORTRAN listing. The code is structured as follows.

First, a 2-D search is done to determine which are the four structural nodes that surround the given aerodynamic grid point. The structural nodes are assumed to form a structured 2-D grid. As mentioned in the code, "the algorithm is to count intersections using the Jordan Curve Theorem and to bisect as in Weierstrass approximation."

After identifying the cell that contains the given aerodynamic point, the actual inverse isoparametric mapping takes place. The code uses bilinear shape functions, *i.e.*

$$\begin{aligned} N_1 &= (1-\xi)(1-\eta) \quad & N_2 &= \xi(1-\eta) \\ N_3 &= (1-\xi)\eta \quad & N_4 &= \xi\eta \end{aligned} \tag{9}$$

which do not follow the classical definition of $\xi, \eta \in [-1, +1]$ but rather $\xi, \eta \in [0, 1]$. For the bilinear case, it is possible to write a closed form solution for (ξ, η) instead of doing the bisection. Expanding Eq. (1) and using Eq. (9), one obtains

$$x_m = a_{x_1}\xi\eta + a_{x_2}\xi + a_{x_3}\eta + a_{x_4} \tag{10}$$

$$y_m = a_{y_1}\xi\eta + a_{y_2}\xi + a_{y_3}\eta + a_{y_4} \tag{11}$$

where

$$\begin{aligned} a_{x_1} &= x_1^e - x_2^e - x_3^e + x_4^e & a_{x_2} &= x_2^e - x_1^e \\ a_{x_3} &= x_3^e - x_1^e & a_{x_4} &= x_1^e \\ a_{y_1} &= y_1^e - y_2^e - y_3^e + y_4^e & a_{y_2} &= y_2^e - y_1^e \\ a_{y_3} &= y_3^e - y_1^e & a_{y_4} &= y_1^e \end{aligned} \tag{12}$$

Eliminating η from Eq. (11)

$$\eta = \frac{y_m - a_{y_2}\xi - a_{y_4}}{a_{y_1}\xi + a_{y_3}} \tag{13}$$

which assumes that $a_{y_1}\xi + a_{y_3} \neq 0$. Substituting this result into Eq. (10), one gets

$$A\xi^2 + B\xi + C = 0 \tag{14}$$

as defined in Eq. (6), with

$$\begin{aligned} A &= a_{x_2}a_{y_1} - a_{x_1}a_{y_2} \\ B &= a_{y_1}(x_m - a_{x_4}) + a_{x_3}a_{y_2} - a_{x_2}a_{y_3} - a_{x_1}(y_m - a_{y_4}) \\ C &= a_{y_3}(x_m - a_{x_4}) - a_{x_3}(y_m - a_{y_4}) \end{aligned} \tag{15}$$

If $a_{y_1}\xi + a_{y_3} = 0$, then one should use Eq. (10) instead of Eq. (11) to isolate η. The same procedure presented above follows from there.

Limitations

The present method is only suitable for interpolation. For regions outside the original structural grid (*e.g.*, control surfaces or any other kind of aerodynamic surface), the structural grid should be extended to enclose them all. Extrapolation can be done with one of the well known linear or quadratic or cubic spline techniques. The information obtained on the extrapolated grid is not as accurate as the one obtained on the original grid. Also, the way the formulation has been presented, it is restricted to a 2-D structural domain. The 3-D formulation is still to be derived.

References

[1] Bathe, K. J., *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1982.

[2] Murti, V. and Valliappan, S., "Numerical Inverse Isoparametric Mapping in Remeshing and Nodal Quantity Contouring," *Computers and Structures*, Vol. 22, No. 6, 1986, pp. 1011 – 1021.

[3] Pidaparti, R. M. V., "Structural and Aerodynamic Data Transformation Using Inverse Isoparametric Mapping," *Journal of Aircraft*, Vol. 29, No. 3, 1992, pp. 507 – 509.

[4] Byun, Chansup and Guruswamy, Guru P., "Wing-body Aeroelasticity Using Finite-Difference Fluid/Finite-Element Structural Equations on Parallel Computers," In *Proceedings of the 35th Structures, Structural Dynamics, and Materials Conference, Hilton Head, South Carolina.* AIAA, April 18 – 20 1994, pp. 1356 – 1365, AIAA Paper No. 94-1487.

[5] Fithen, R., "INTERP," Private correspondence, 1995.

10. DESCRIPTION OF THE ANALYTICAL TEST CASES

The analytical or mathematical test cases were designed to examine a number of different situations that could be encountered during use of the algorithms. Functions were chosen to represent the different types of data which might arise during modal analysis, loads integration, etc. The purpose of these tests were to determine the limiting characteristics of each of the algorithms.

Therefore, each algorithm was examined over a broad range of functions. In addition, other factors that needed to be considered included: directional bias, sensitivity to amplitude, sensitivity to extrapolation, and combinations of complex (sinusoidal) and simple (constant) functions superimposed in different directions.

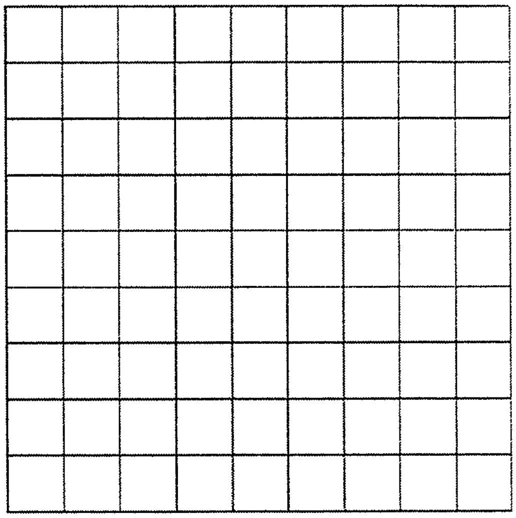
The limiting or most inaccurate test cases are those in which the known data are provided on a sparse grid and must be converted to a finer grid. This correlates (most often) to the conversion of data from a structural (CSD) to an aerodynamic (CFD) grid. Thus, all of the known function grids are sparse and include both regular (equally spaced) and irregular (clustered or non-equally spaced) grids. Examples of these grids are shown in Figure 10-1a)–e). Figure 10-1a) is the sparsest grid used, a 10×10 plate. To ensure that no coding errors existed, test cases were interpolated to an identical grid. Errors should be on the order of machine accuracy if the algorithms are correctly coded. Figure 10-1b) depicts one of the finer initial grids that was used to interface more complex (multiple sinusoidal) functions. Figure 10-1c) is the shell counterpart grid to the plate grid shown in Figure 10-1a), just as Figure 10-1d) is the shell counterpart to Figure 10-1b).

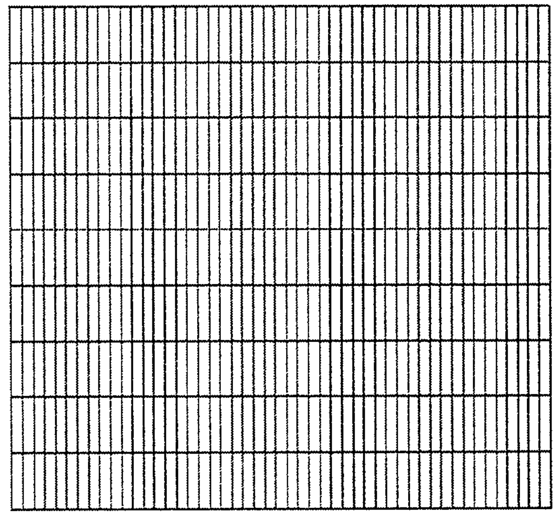
To ensure that the methodologies include no bias for irregular grids, a series of test cases was developed for an irregular grid, such as those that may be encountered in some finite element models and experimental configurations. Figure 10-1e) depicts one of these irregular grids.

In order to examine all of these factors, a series of five test sets was developed. These test sets included over 260 test cases, from constant functions to 7×7 cycle oscillations. A list of these test cases, with the functions that were interpolated along with the details of the grids associated with the given and interpolated functions are provided in Appendix B. Constant functions, such as those found in near-constant mode shapes and separated pressure distributions were examined at levels of 5 and 50 to check dependency on function magnitude. Linear functions, found in mode shapes, structural matrices, pressure distributions, deflection and slope values, were varied both in overall magnitude (5, 1, .01) and in difference magnitude ($\Delta\pm5$, $\Delta\pm.01$). Sinusoidal functions are also integral components in all types of data to be examined. The peak-to-peak amplitude was varied (2 and .02), as well as the frequency (1, 3, 5, 7 cycles). The higher cycles were examined specifically as limiting cases that might be encountered during slope interpolations. Examples of the contours describing several of the sample cases are provided in Figure 10-2.

For each of the five test sets, specific characteristics were examined for both plate and shell-like grids. These test sets are summarized as:

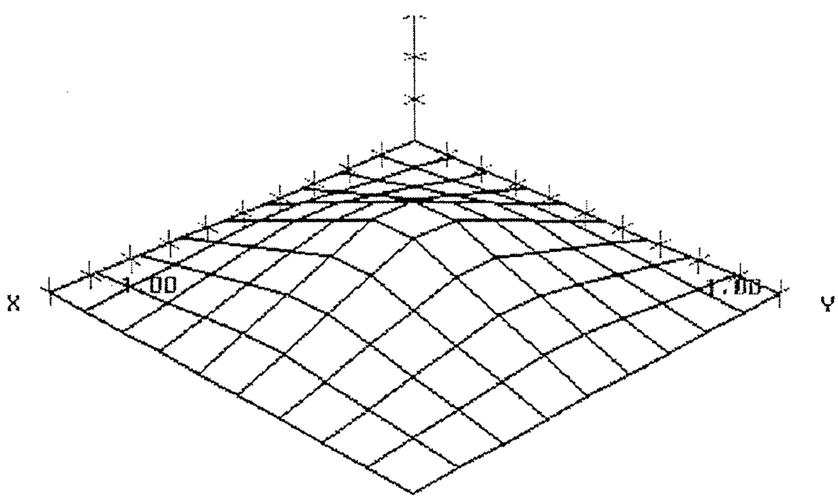
Test Set	Characteristics Examined
1	a) algorithm check, b) directional bias c) regular, structured grid interpolation
2	irregular grid interpolation
3	extrapolation (regular and irregular grids)
4	diminishing variation
5	beam element model





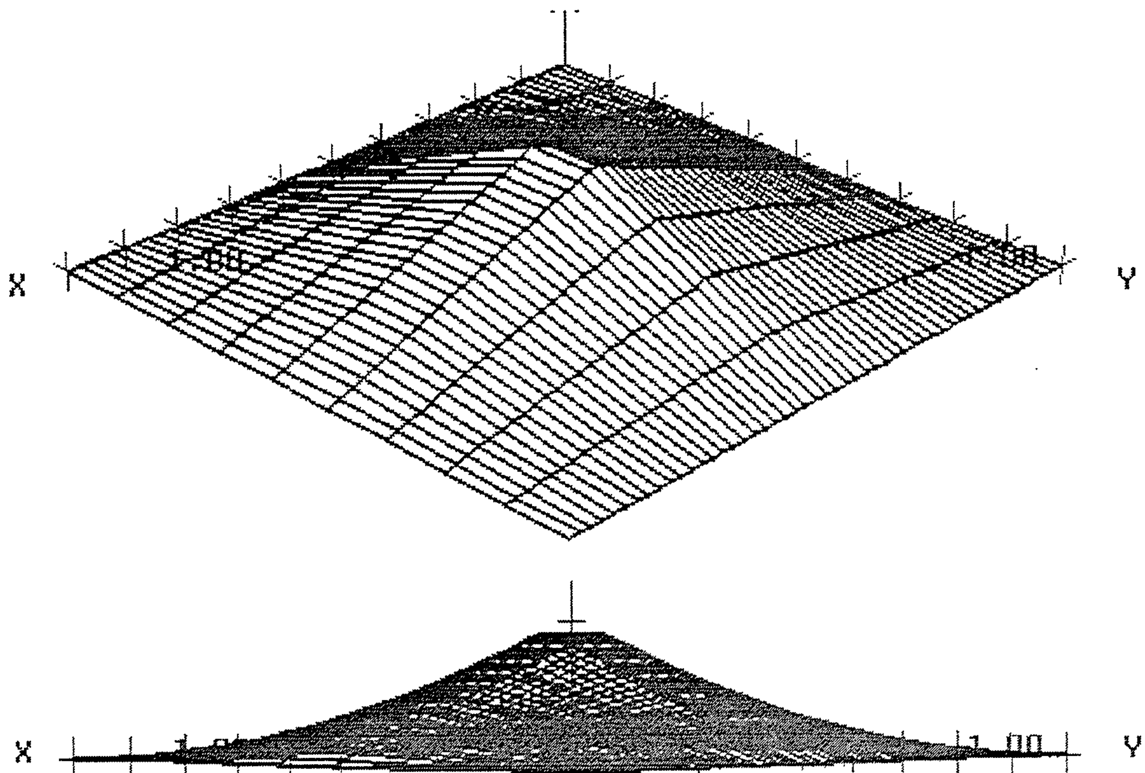
a) 10 x 10 Regular Grid for Known and Interpolated Functions

b) 50 x 10 Grid for Known Functions

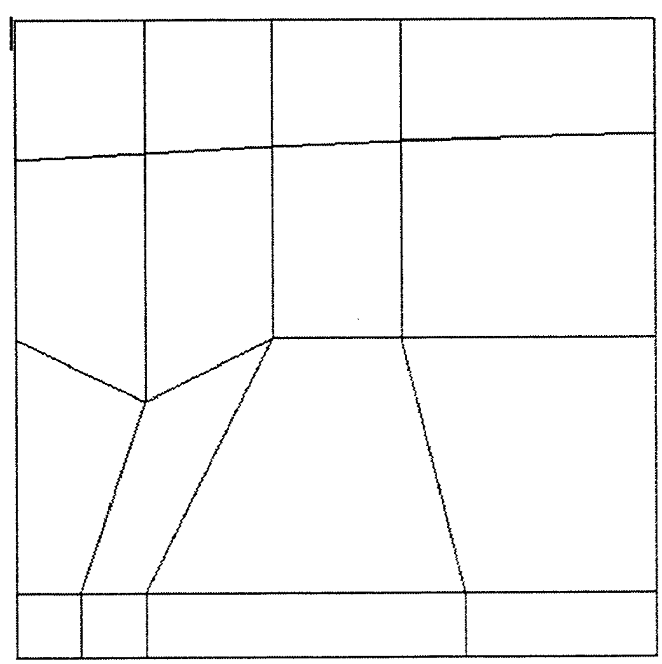


c) 10 x 10 Regular Shell Grid for Known and Unknown Functions

Figure 10-1. A Sampling of the Grids Used for the Mathematical Formulation

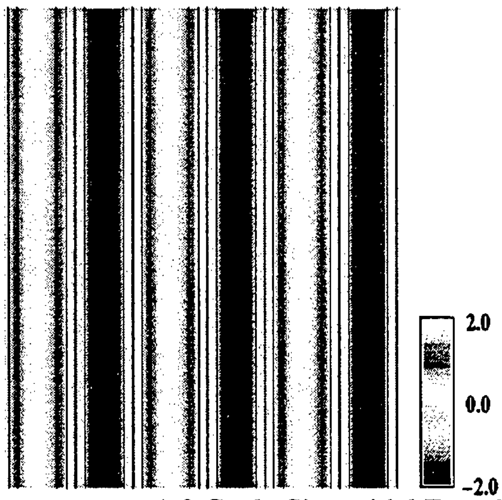


d) 50 x 10 Regular Shell Grid for Known and Unknown Functions

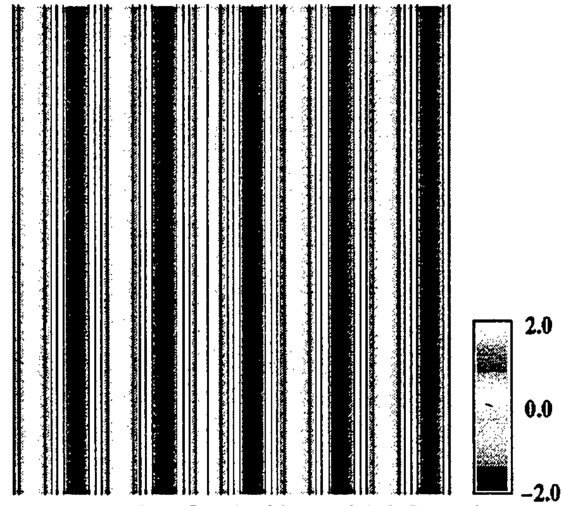


e) Grid 1 for Test Set 2 (Known Functions)

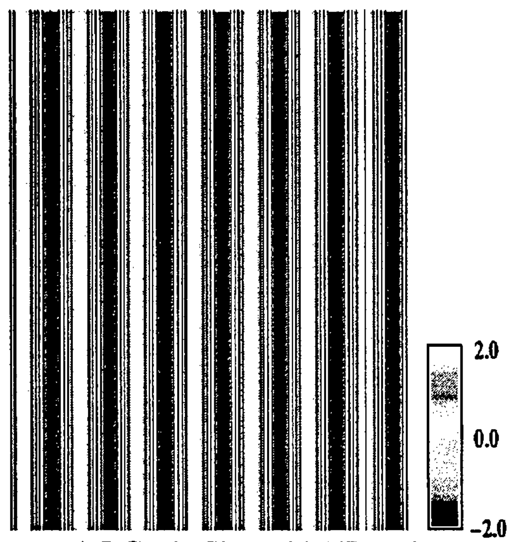
Figure 10-1. A Sampling of the Grids Used for the Mathematical Formulation (concluded).



a) 3 Cycle Sinusoidal Function

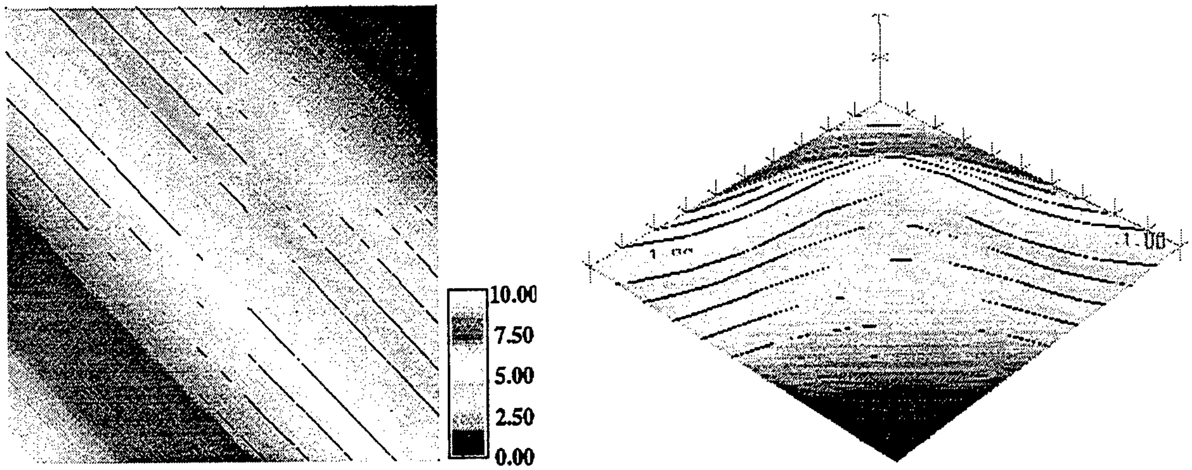


b) 5 Cycle Sinusoidal Function

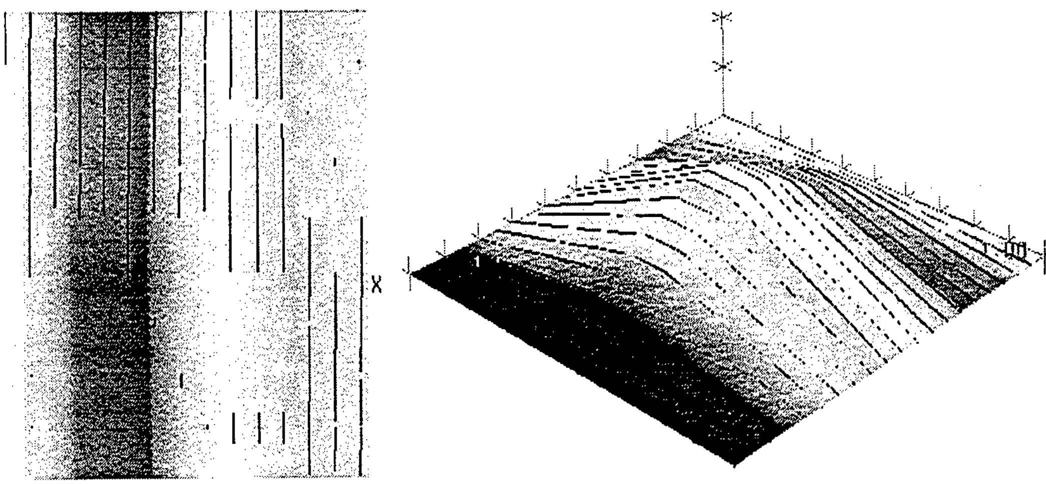


c) 7 Cycle Sinusoidal Function

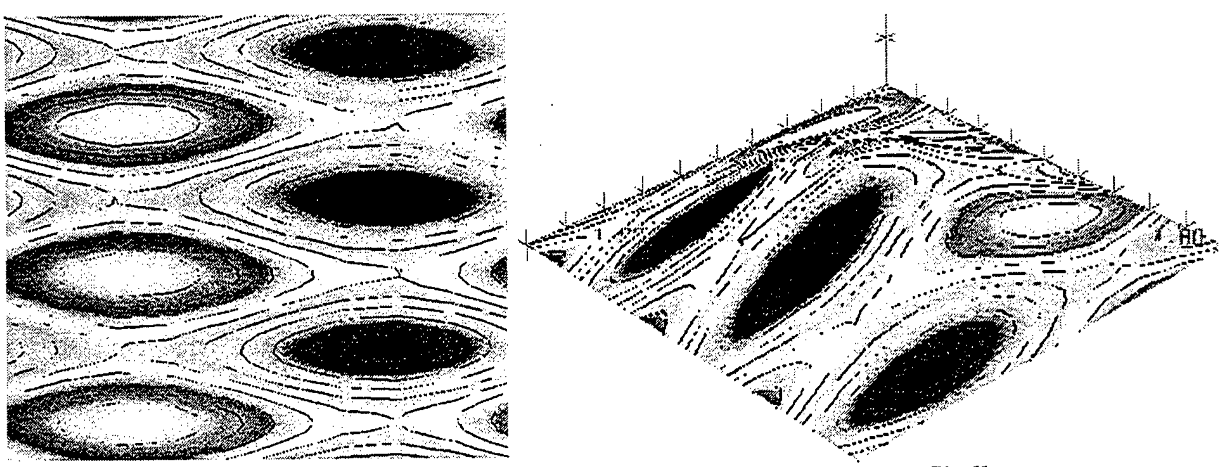
Figure 10-2. A Sampling of the Mathematical Functions Evaluated During the Test Analysis.



d) Linearly Varying Functions in Both Directions for a Shell



e) Linearly Varying Function in One Direction for a Shell



f) One Cycle by Three Cycle Sinusoidal Function on a Shell

Figure 10-2. A Sampling of the Mathematical Functions Evaluated During the Test Analysis. (Concluded)

Test Set 1 – A parametric study of various functions representing the level and shape of the types of data that may be applied in aeroelastic applications were examined for a series of two- and three-dimensional grids which represent typical structured (quadrilateral element) structural and CFD grid topologies. The impact of grid clustering was also examined here. The sensitivity of the algorithm to the direction of interpolation over different directions (i.e., functions that vary over the spanwise direction or the streamwise direction) was also examined. This set of data consisted of 138 test cases. The first subset examined the two-dimensional characteristics for streamwise functions. These functions included: constant surfaces (A=5., 50.), linear functions (A=5. – 0., 1.01 – 1.02, 0.01 – 0.02), and sinusoidal functions (A=2, 1 cycle; A=2, 3 cycles; A=2, 5 cycles; A=2, 7 cycles; A=0.02, 3 cycles; A=0.02, 5 cycles; A=0.02, 7 cycles).

The second subset examined the two-dimensional characteristics described in subset 1 for spanwise functions. The third, fourth and fifth subsets examined these functions for plates and shells, using superposition of these functions to examine more realistic surface data.

Test Set 2 – A parametric study of various functions representing the level and shape of the types of data that may be applied in aeroelastic applications were examined for a series of two- and three-dimensional grids that represent typical irregular structural (e.g., experimental data) and unstructured (irregular, mixed triangular and quadrilateral) CFD grid topologies. This set of data consisted of 28 test cases. Functions similar to those in test set 1 were examined. The amplitudes of the functions were decreased by another order of magnitude to examine those effects.

Test Set 3 – Using grid topologies and functions from Test Sets 1 and 2, the methods were examined for their abilities to perform extrapolations, such as those that may be required for unevenly matched topologies. This test set consisted of 61 test cases for both shell and plate configurations. The functions were similar to those described in test sets 1 and 2.

Test Set 4 – Using grid topologies and functions from Test Sets 1 and 2, the integrity of the methods is examined by interpolation to finer and finer grids. The presence of numerical oscillations generated in these tests indicate potential problems when converting to very fine grids or when using the methodology for multi-grid or grid sequencing CFD codes. This data set consisted of 30 test cases, with 2 higher cycles above the initial cycle (computed in test set 1, runs A – J).

Test Set 5 - The test cases for test set 5 represent a cantilever beam of unit length. The beam is originally along the x-axis, and every deflection occurs in the z direction. Two deflections are considered: a static deflection (bending) due to a distributed force and the first three natural vibration (bending) modes. The first subset of cases (test5a – r) has no static deflection, only the three bending modes, with two levels of maximum amplitude (test5a – i, A=1.0; test5j – r, A=0.1), and three different grids (10×10; 100×100; 100×500) forming a regular grid. The second set of cases (test5a1 – r1) has similar structure as the first subset, but now there is a superposition of a static deflection (test5a1 – i1, $A_{static} = 1.0$; test5j1 – r1, $A_{static} = 0.1$).

One of the primary purposes of using these sets of known functions was that a set of analytical solutions can be computed for direct comparison. During the initial examination of these mathematical test cases, the interfaced (interpolated) data were compared with data analytically computed from the original functions at the new grid locations. While these data provided good correlation data for the simpler functions, it was discovered that these data introduced artificially large errors for the sinusoidal functions. These errors were inflated because the original functions did not include some of the function features that a finer grid would be able to capture, as shown for a three-cycle sinusoid in Figure 10-3. To alleviate this problem, a linear interpolation of the analytical data at points on the known grid to points on the unknown grid was used. While these

data did introduce slight errors due to the nature of the interpolation, it provided a much more accurate error assessment for the sinusoidal functions. When large errors were reported, each interpolated function was visually examined and compared with the original, linear interpolated, and analytical functions.

Each of the analytical test categories was examined visually through the use of the commercial graphics routines by overlaying the original and interpolated functions. Carpet plots of the errors between the original and interpolated functions exhibit the presence of numerically-induced oscillations which, in geometry updates or loads computations, can result in non-physical behavior in CFD or CSD simulations. In addition, a quantitative analysis was computed which tabulated the errors from a statistical viewpoint through maximum errors (and locations), averages and standard deviations. These quantitative results were computed using the following equations:

Maximum Error (Absolute) :

$$\text{Maximum of } \left(\text{Function}_{\text{Interpolated}} - \text{Function}_{\text{Computed}}\right) \text{ for all points}$$

Maximum Error (Percent) : $$\frac{\text{Maximum Error}}{(\text{Maximum Computed Function Value})} \times 100.$$

Average Error (Absolute) : $$\frac{\sum\left(\text{Function}_{\text{Interpolated}} - \text{Function}_{\text{Computed}}\right)}{(\text{Number of Points - 1})}$$

Average Error (Percent) : $$\frac{\text{Average Error}}{(\text{Maximum Computed Function Value})} \times 100.$$

Standard Deviation : $$\sqrt{\frac{\sum\left(\text{Error} - \text{Error}_{\text{Average}}\right)^2}{(\text{Number of Points - 1})}}$$

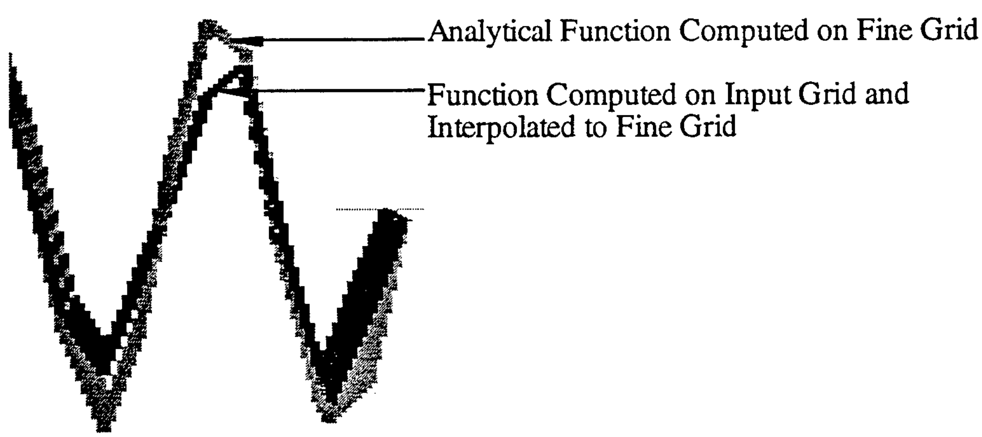


Figure 10-3. Comparison of the Differences in the Function Values for a Fine Grid Using Analytical and Interpolated (from Coarse Grid) Methods to Compute the Function Values

11. RESULTS OF THE ANALYTICAL TEST CASES

The analytical test cases discussed in Chapter 10 were examined, using each of the methods of Chapters 4–9. Results from these test cases are discussed in detail in the following sections. Each section is self-contained, including all its associated tables and figures along with the text. Plots are presented using FAST [7] and TECPLOT [8].

Statistical summary plots of the test cases are included in the following subsections. These summary plots provide visual correlation of the results for each subsection. The interpolation errors are plotted as relative errors. That is the difference in the original and interpolated function values are divided by the maximum value of the original function, and multiplied by 100 to form a percentage error. These numbers correlate with the Percentage Errors of the tabulated data included in this chapter. Absolute errors are also listed in the tabular data, but are not plotted.

The results for each method are discussed in several subsections which address specific aspects of the interpolation results. These subsections and their topics include :

Overall Accuracy – Statistical performance of the method for all test cases is discussed.

Grid Spacing Sensitivity – For all of the test cases reviewed for this topic, a function was defined on a regular, uniform grid. The function was then interpolated onto grids which were regular or included clustering in different directions. The interpolations for the different grids are then compared. These cases are important since they are analogous to the transformation of mode shapes from a coarse structural grid to a finely clustered aerodynamic (CFD) grid.

Directional Bias – The test cases reviewed in this subsection ensure that the interpolation method can handle complex functions which vary in both surface directions (hereafter referred to as the streamwise and spanwise directions). Directional bias is indicated if the method does not provide identical interpolations when the grids and functions are rotated ninety degrees to lie in the other coordinate direction.

Magnitude/Amplitude Sensitivity – Test cases for functions with different amplitudes and magnitudes are compared to determine if the method is sensitive to overall function levels. This is important since a wide variety of parameters will be transformed using these methods. Sensitivity to magnitude indicates the need for parameterization of the function. For comparison, plots are included where the order of the magnitude or amplitude is the abscissa. For example, functions with magnitudes of 1, 2, or 5 are all plotted as an order of magnitude of 1. Similarly, 0.01, 0.02 and 0.05 are plotted as an order of magnitude of 0.01. By plotting the order of the magnitudes, rather than the actual magnitudes, consistent comparisons can be drawn.

Frequency Sensitivity – Test cases are examined with functions that increase in frequency (number of cycles increases over the surface). These higher order oscillation results illustrate the impact of interpolating higher order mode shapes as well as pressure distributions.

Extrapolation – Test set 3 is designed to examine the algorithm's ability to extrapolate data. Extrapolation is very important since it is unlikely that the structural and aerodynamic grids will be coincident. Extrapolation may be required in all three Cartesian directions, which means that thickness may also be required to be extrapolated. In these tests, thickness corresponds to the z (normal) direction. The extrapolation function test cases 3A – 3H1 correspond with the test cases 1A – 1H1 and 4A – 4H1 (the normal direction is computed differently in test case 4, but does lead

to identical functions). The difference between test sets 1 (and 4) and 3 is that the starting grids (known functions) for test set 3 do not extend to leading and trailing edges of the unknown function grids in either direction. This corresponds to a structural grid which does not have points at the leading and trailing edges, the root and tip of an aerodynamic wing grid.

Diminishing Variation – The interpolation of the grid should be consistent no matter how fine the grid becomes. Thus, a series of three grids with increasing fineness was examined. The original grid was taken from test set 1 (A – H, A1 – H1, and A2 – H2). The two finer grids were computed by doubling the number of points in each direction. If the errors change, this is indicative that oscillations are being introduced into the interpolation as the grid becomes finer.

Sensitivity to Three-Dimensional Surfaces (Plates vs. Shells) – The various test sets include flat surfaces (plates) and surfaces which vary in the normal direction (shells). This subsection indicates if the method, as implemented, has sensitivities associated with the interpolation.

Algorithm CPU Memory and Time Requirements – An algorithm to compute the required CPU memory for a given set of parameters is given. The time requirements based on the actual runs are provided as a statistic.

11.1 Infinite-Plate Spline Method (IPS)

The IPS method is the method most commonly used in the interpolation of data from structural and aerodynamic grids (See Chapter 3). This methodology is of particular interest since it is applied for so many different applications.

The implementation of IPS is described in Chapter 4. Shan [6] discovered that the method performs best with Singular Value Decomposition (SVD). This is the implementation applied during this study. Using SVD, it is possible for the user to modify the zero threshold; i.e., any values below the zero threshold are considered to be zero during the matrix decomposition. Unfortunately, this adds an additional input for the user to understand and apply. Indeed, Shan discovered that without a judicious selection of the zero threshold, the results for a single problem can vary dramatically. Thus, the question is raised about the validity of the SVD scheme. Without applying SVD – and its ability to change the thresholds – the matrix can be ill-conditioned or singular in many instances where other methodologies (discussed later in this chapter) have no problem. This is particularly prevalent for the higher frequencies of the sinusoidal inputs. In this research, IPS was first run with a threshold of zero (all points used in the matrix solution).

Overall Accuracy – Statistical summaries of all test cases run with IPS are presented in Tables 11.1 to 11.5. The overall accuracy of IPS is adequate, but not consistent across the range of test case functions. This is demonstrated in Figures 11-1 and 11-2 where the relative error is plotted for each category of function. It is easily observed from Figures 11-1 and 11-2 that for constants and some linear and sinusoidal functions the method has excellent performance. For many of the functions, however, IPS produces in large errors, as seen by the statistical summaries in Tables 11.1 – 11.5. This is particularly true of sinusoidal functions. For most of the runs there is not a large difference between the maximum and average errors. This indicates that the method is not sensitive to any particular location.

Grid Spacing Sensitivity – Figures 11-1 and 11-2 indicate that IPS is sensitive to grid spacing. The sets of runs marked "A" on the figures indicate interpolation to an identical grid. For the sets of data marked with "B" the function was interpolated to finer, clustered grids. The error increased between 8 to 12 orders of magnitude between these two types of grids. The highest errors constituted a range of 4% to over 100% (based on the function magnitude).

Additionally, the interpolated data is shown to be oscillatory in nature. These oscillations are depicted in Figure 11-3, a typical interpolation error plot by IPS. The impact of these oscillations can be felt for interpolation of grid deflections from the structural to the aerodynamic grid. Oscillations in the updated surface grid of a wing can result in non-physical pressure distributions, and can cause flow separation bubbles or transition to turbulent flow where laminar flow actually occurs. These oscillations can be smoothed out using several different smoothing algorithms, as described by Shan [6]. These smoothing algorithms act as a low-pass filter, removing the higher oscillations from the interpolation. However, these smoothing algorithms have been shown to detrimentally impact the accuracy of the interpolation. Since the use of smoothing algorithms is very user-intensive and must be confirmed graphically for each interpolated curve, it is not recommended for this application.

Directional Bias – Figure 11-1 includes the interpolations of functions for both the streamwise (test1a – 11) and spanwise (test1a2 – 12) directions. These interpolations are virtually identical, indicating IPS has little or no sensitivity to the direction of the function.

Magnitude/Amplitude Sensitivity – The IPS magnitude study results are plotted in Figure 11-4 for test set 1. For each of the different combinations of grids and functions, the relative errors of the interpolation are constant as the function amplitudes and/or magnitudes change. This trend is consistent for function type, plates, shells, direction of the function and grid (test sets 1 and 2). Thus, IPS is not sensitive to magnitude changes, as expected since it is a linear method.

Sensitivity to Frequency (Higher Oscillations) – IPS is very sensitive to the frequency of the function. The sinusoidal functions shown in Figures 11-1 and 11-2 are for one cycle over the surface, as illustrated in Figure 11-3. When the frequency of the sinusoidal function is increased to three, five, and seven cycles, the interpolation errors increase rapidly, as seen in Figures 11-5 for test set 1. For these higher oscillations, the number of points in the direction of the function was proportionally increased so that the interpolating algorithm was provided an adequate number of input points from which to perform the interpolations. A variation of 8 to 13 orders of magnitude is seen when the function increases from one to three cycles. The large variation in error is not seen during the increase from three to five cycles. Note, however, that by the three-cycle function, the magnitude of the relative error (50 – 100%) has reached the same order of magnitude as the amplitude of the function. IPS interpolation failed for sinusoidal functions greater than five cycles.

The reason for the large jump in error between one and five cycles is seen in Figure 11-6. Here, the interpolation result for a three-cycle sinusoid is plotted. Notice that while the function amplitude is captured along the surface edges, the amplitude prediction has significantly degraded along the interior of the grid. The oscillations discussed earlier, and plotted for a one-cycle sinusoid oscillation in Figure 11-3 are now dramatically apparent. These oscillations occur in the direction which is not rapidly varying, indicating a cross-coupling with the highest function frequency.

Extrapolation – The maximum errors for test sets 1 and 3 are plotted in Figure 11-7 to illustrate the impact of extrapolation versus interpolation. It is apparent from this figure that IPS is not as accurate for extrapolations since the relative errors increase by approximately an order of magnitude or more for functions which include a sinusoidal component. The cases which are comprised of only constants and linearly varying components do not, for most cases, have any problems in extrapolation. However, it must be realized that most pressures and mode shapes will rapidly vary in the regions which are most likely to require extrapolation (leading edge, trailing edge, root, tip, etc.). Thus the inability of IPS to adequately resolve these data is important. The percentage of the relative errors are now between 10% and 100% for extrapolation. The location of the maximum error has moved, in most instances, to the region of extrapolation (refer to Table 11.3). The presence of the large error at the edges of the plate where extrapolation is occurring has been reported by several authors [9, 10, 6]. These errors are characterized as the "potato chip" effect because of the large curvature which is introduced by IPS. An example of this error is shown in Figure 11-8.

Diminishing Variation – Figure 11-9, Table 11.1 and Table 11.4 show that the interpolation error either decreases or does not vary with increasing grid fineness, indicating that the interpolation scheme is consistent for each function. If the errors increase, this indicates that oscillations are being introduced into the interpolation as the grid becomes finer. The typical trend for IPS is a drop in interpolation error from the first grid doubling, and minimal change in interpolation error for increases in grid fineness thereafter.

Sensitivity to Three-Dimensional Surfaces (Plates Vs. Shells) – From Figures 11-1 to 11-9, the only sensitivity that IPS has when a plate is extended to a shell is for the constant and linear functions. (The two-dimensional equations described in Chapter 4 have been expanded to

encompass three-dimensional surfaces using arclengths.) The sensitivity to sinusoidal functions is minimal.

Algorithm CPU Memory and Time Requirements – The average CPU time requirement is approximately 10 – 12 seconds (refer to the last column of Tables 11.1 through 11.4, with the exception of test cases that have very large errors, indicative of ill-conditioned matrices. These runs require an average of 1400 to 1700 seconds. Matrices that never converge can require over 3600 seconds before they stop with an error flag.

The algorithm CPU memory requirement can be computed using the following algorithm:

$$\text{CPU SIZE (in MegaBytes)} = 0.3034 + 2.1902\times10^{-4}\text{KGS} + 1.6\times10^{-5}\text{KGS}^2 + 5.6988\times10^{-5}\text{UKS} - 7.9967\times10^{-11}\text{UKS}^2$$

where KGS is the total number of points for the grid where the function is known and UKS is the total number of mesh points for the grid to which the function is to be interpolated. For example, if mode shapes are to be interpolated from a CSD mesh to a CFD mesh, KGS is the number of mesh points for the CSD mesh, and UKS is the number of surface CFD grid points. This function yields values within 1% of the actual CPU size during testing. Figure 11-10 indicates that the predominant memory requirement is directly related to the number of mesh points for the known function grid. This is as expected since an increase in these mesh points results in a larger matrix to solve.

Single Precision – The obvious impact of reducing the precision of the method from double to single precision is the reduction of the CPU memory requirements. For IPS, this precision reduction results in a savings of approximately 25 – 30%. The CPU time increased by approximately a factor of 2 because of the additional time required to resolve the matrices. For most instances, the maximum error increases, but not to a significant degree. However, very accurate interpolations with errors of the order of 10^{-13} are no longer possible. The maximum accuracy (for both the maximum error and average error) achieved by the single precision method is 10^{-4}. However, for errors above 1% of the maximum magnitude or amplitude, little difference is seen in the interpolation accuracy, as seen in Table 11-5. Thus, there appears to be no reason to use double precision for IPS.

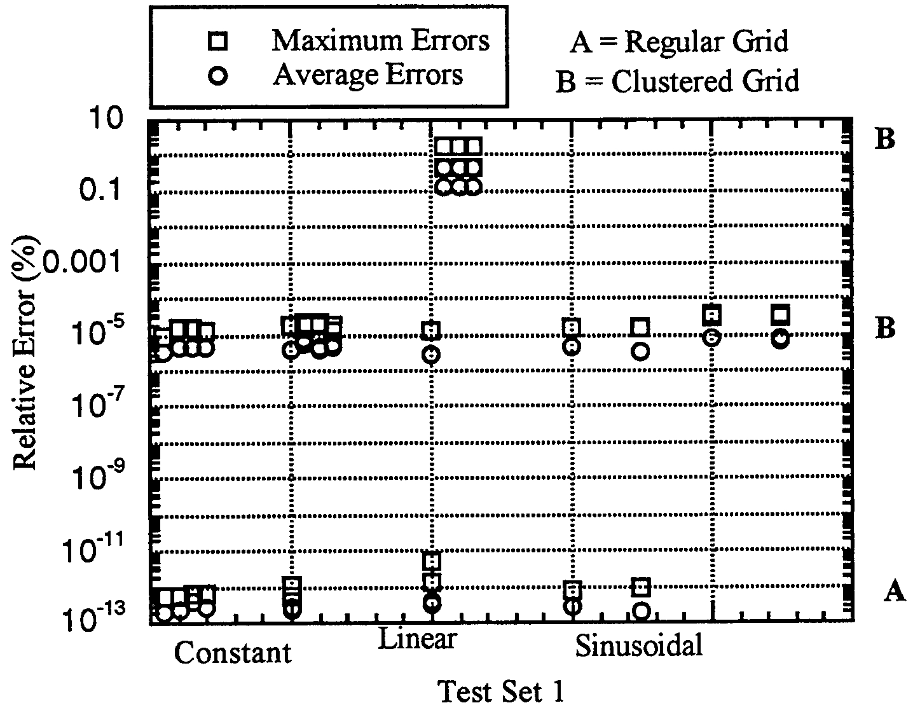


Figure 11-1. Variation of Error for Test Set One Based Upon Function Type for Plates Using the Infinite-Plate Spline Method

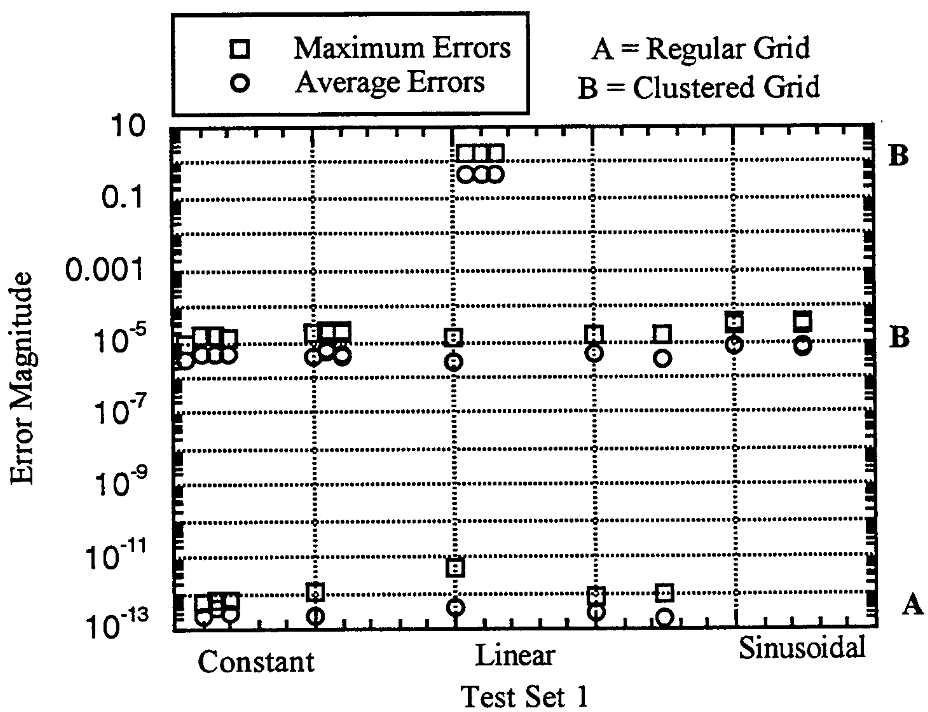
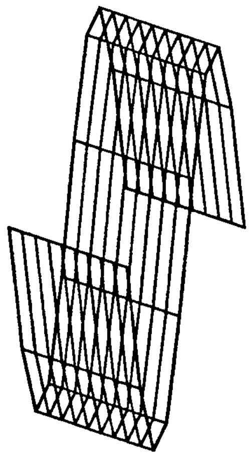
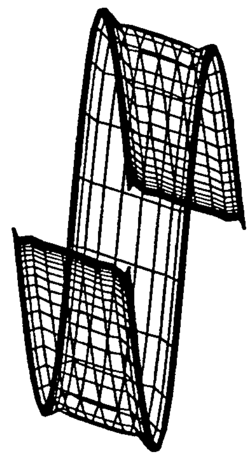


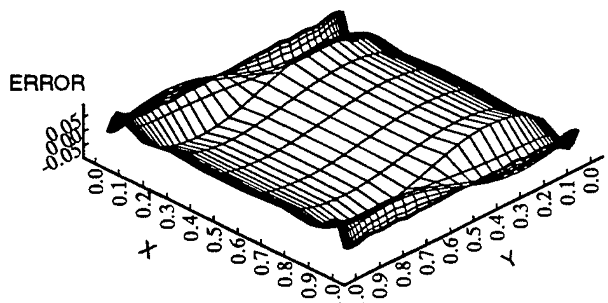
Figure 11-2. Variation of Error for Test Set One Based Upon Function Type for Shells Using the Infinite-Plate Spline Method



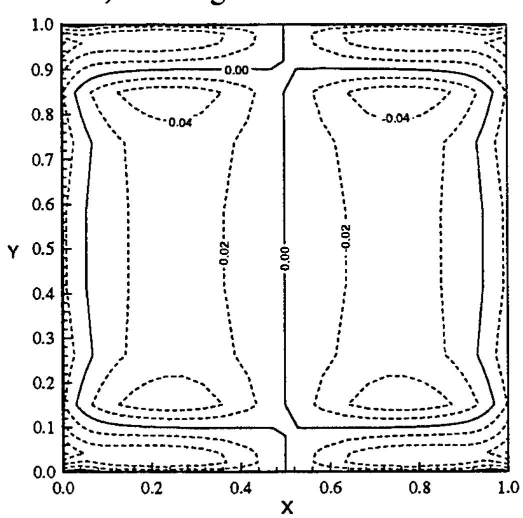
a) Original Function



b) Interpolated Function



c) Orthogonal View of the Error



d) Error Contours

Figure 11-3. Example of Oscillations Induced by the Infinite-Plate Spline Method (Test 1) for a One-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2

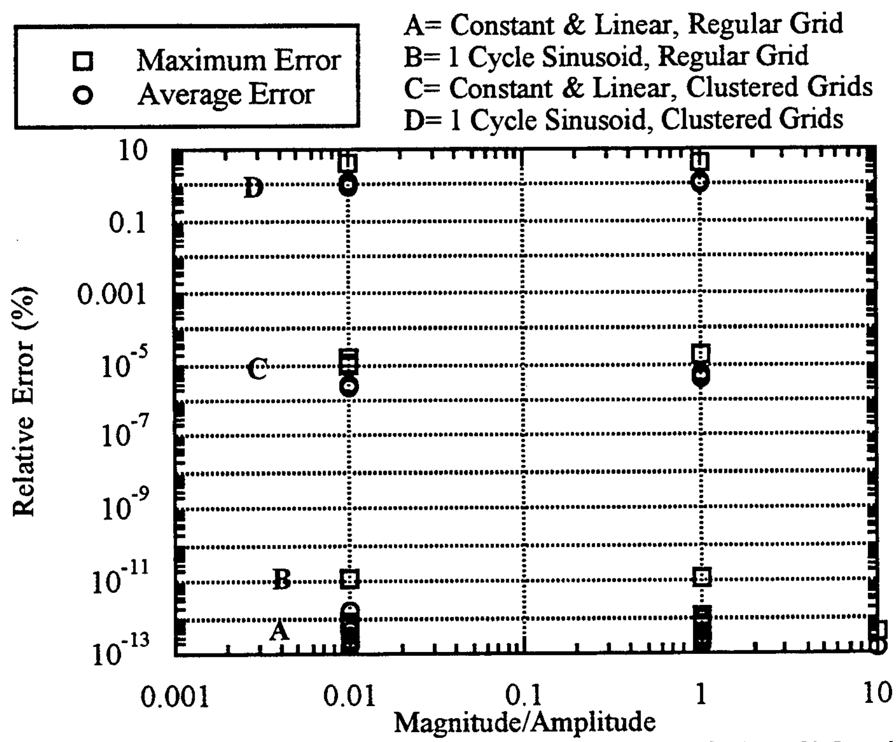


Figure 11-4. Variation of Error for Test Set One Based Upon the Order of Magnitude of the Function Using the Infinite-Plate Spline Method

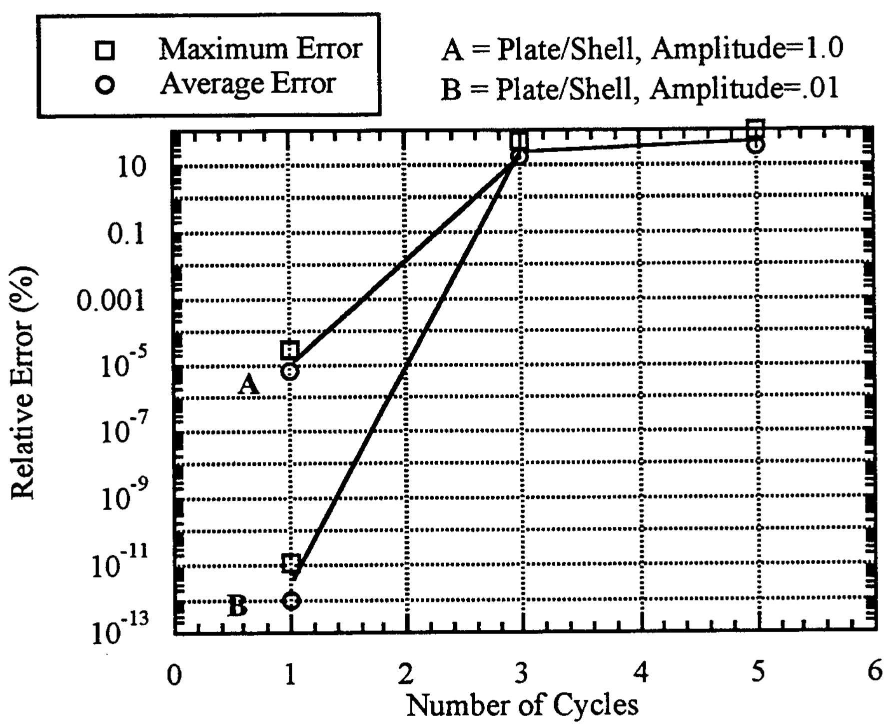
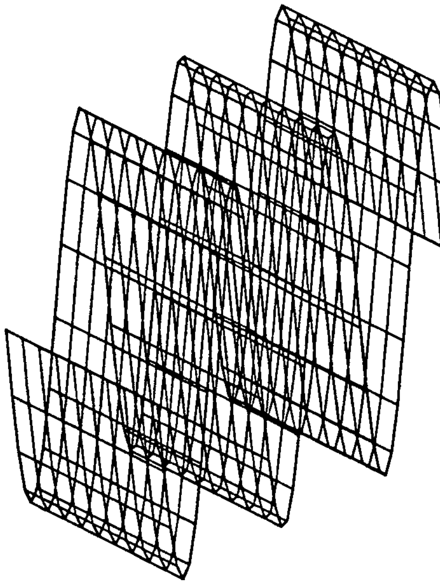
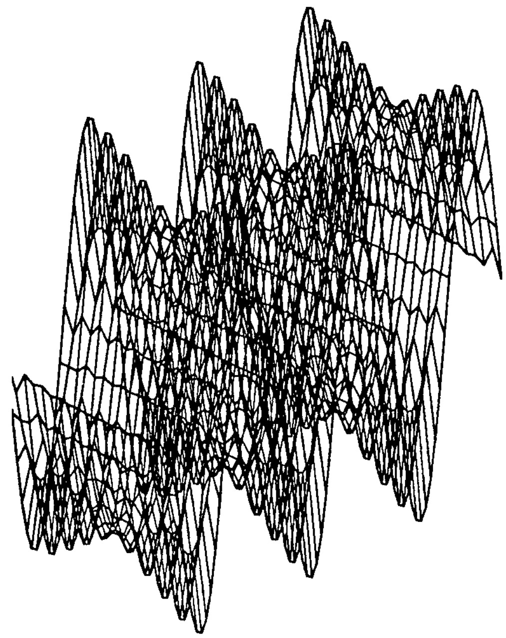


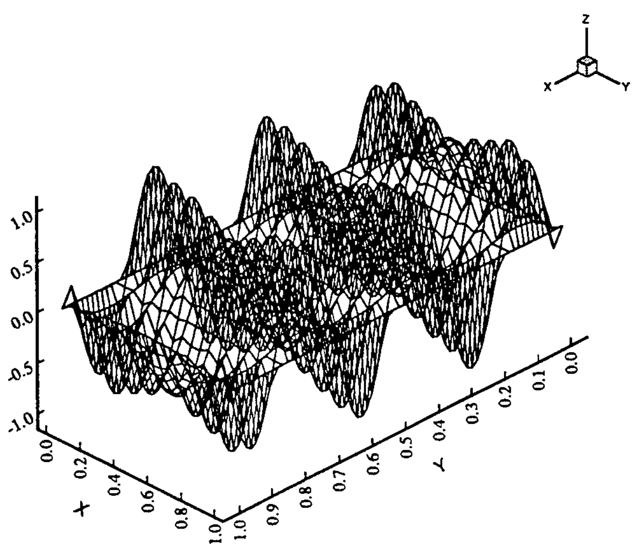
Figure 11-5. Variation of Error for Test Set One Based Upon the Number of Sinusoidal Cycles Using the Infinite-Plate Spline Method



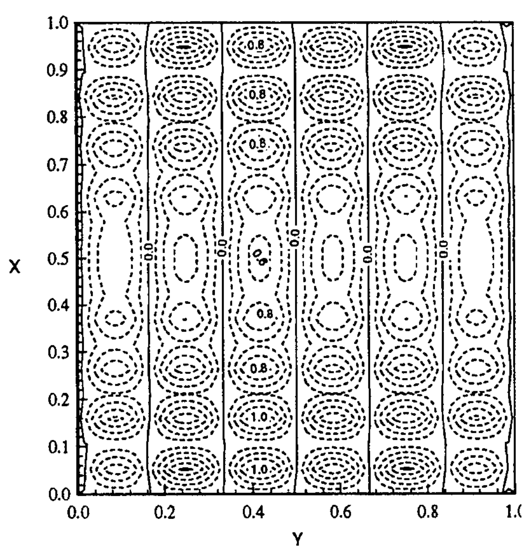
a) Original Function



b) Interpolated Function



c) Orthogonal View of Error



d) Error Contours

Figure 11-6. Example of Oscillations Induced by the Infinite-Plate Spline Method (Test p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2
(X-axis and Y-axis have been expanded for visibility)

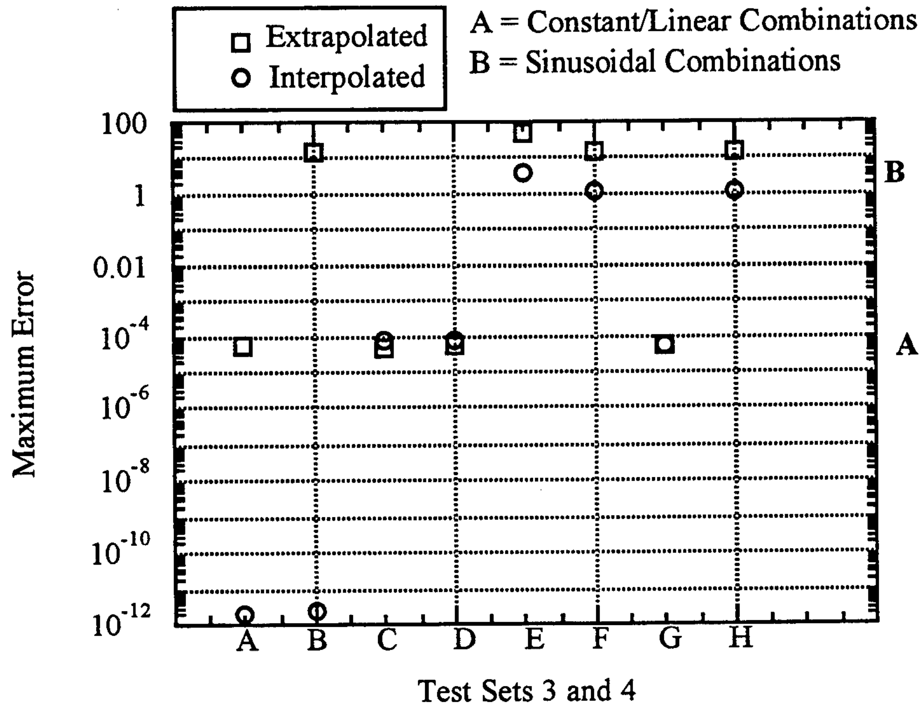


Figure 11-7. Variation of Error for Interpolation (Test Set 1) and Extrapolation (Test Set 3) Using the Infinite-Plate Spline Method

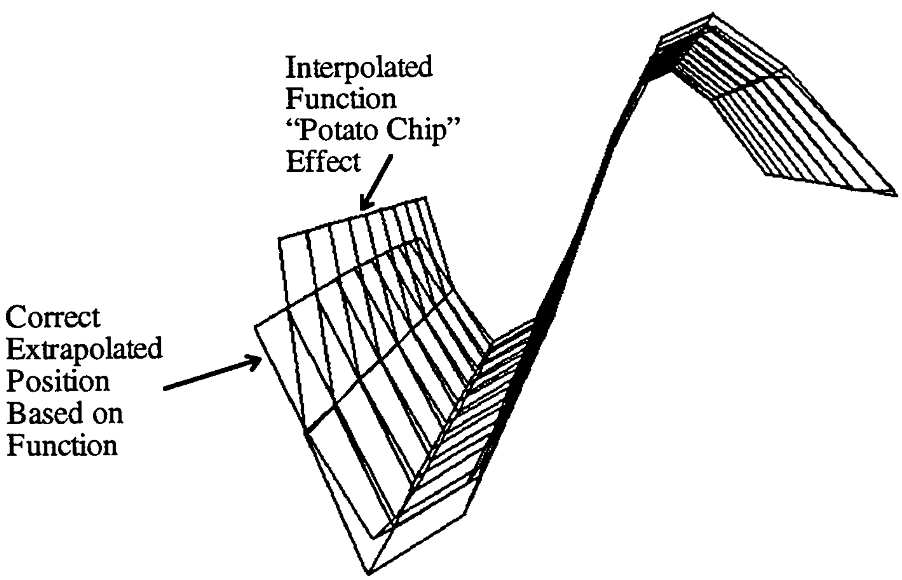


Figure 11-8. An illustration of the "potato chip" extrapolation effect encountered in the Infinite-Plate Spline Method

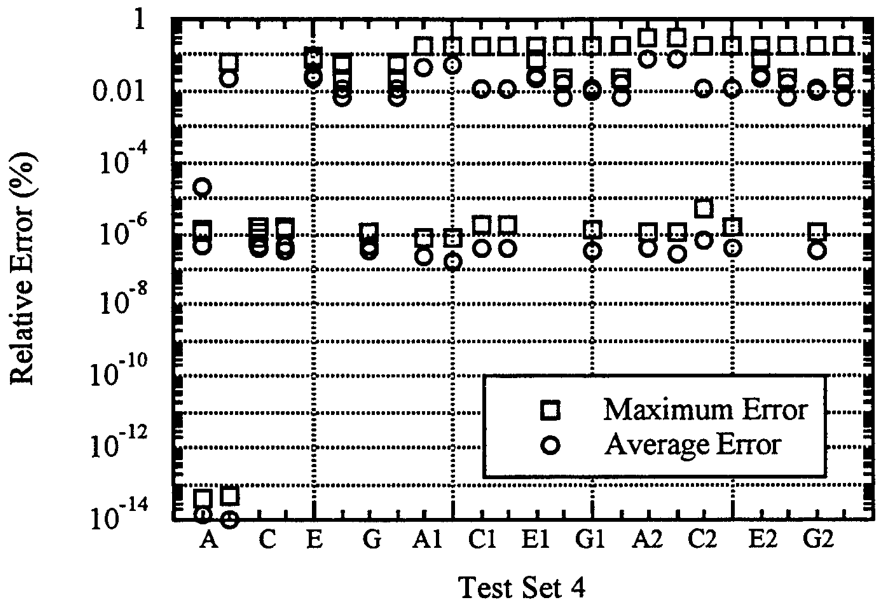


Figure 11-9. Variation of Error for Increasing Grid Fineness (Test Set 1 and 4) Using the Infinite-Plate Spline Method

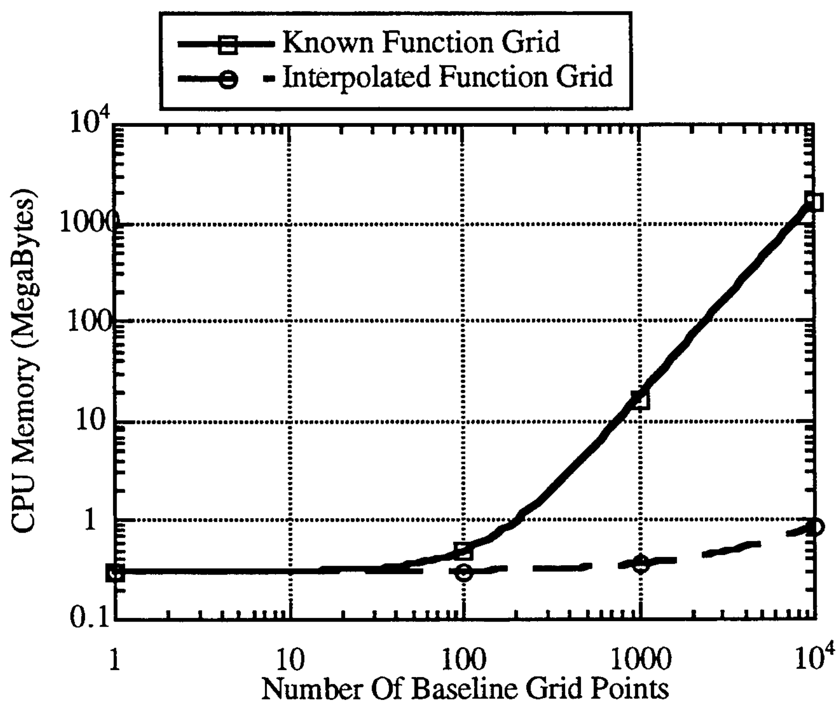


Figure 11-10. CPU Memory Requirements for the Infinite-Plate Spline Method as Implemented with SVD and Threshold Modification

Table 11.1 Statistical Summary of Test Set 1 using Infinite-Plate Spline

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	3.02E-14	6.04E-13	10	10	1.07E-14	2.14E-13	8.30E-15	12.705
test1b	5.11E-14	1.02E-12	3	1	1.10E-14	2.19E-13	2.07E-14	11.58156
test1c	2.42E-13	1.23E-11	1	1	1.84E-14	9.32E-13	3.77E-14	11.32234
test1d	3.02E-14	6.04E-13	50	20	1.23E-14	2.46E-13	5.88E-15	12.14445
test1e	1.01E-06	2.01E-05	27	6	2.94E-07	5.88E-06	5.39E-07	12.01813
test1f	8.53E-02	4.273	50	11	2.31E-02	1.159	4.50E-02	11.89129
test1g	3.11E-14	6.22E-13	50	17	1.86E-14	3.71E-13	1.43E-03	11.8736
test1h	9.07E-07	1.81E-05	12	18	2.04E-07	4.07E-06	4.51E-05	11.71606
test1i	8.53E-02	4.273	50	10	2.31E-02	1.159	3.03E-02	11.65424
test1j	3.20E-14	6.40E-13	3	13	1.45E-14	2.91E-13	9.60E-04	11.69078
test1k	9.79E-07	1.96E-05	22	19	2.49E-07	4.98E-06	3.04E-05	11.65704
test1l	8.36E-02	4.21	50	6	2.40E-02	1.209	3.95E-02	11.4933
test1m	1.99E-13	3.98E-13	1	10	7.16E-14	1.43E-13	3.97E-03	10.62021
test1n	5.77E-15	5.66E-13	1	9	1.86E-15	1.82E-13	3.99E-04	10.57178
test1o	1.23E-16	6.16E-13	2	1	4.76E-17	2.38E-13	4.01E-05	10.58344
test1p	1.045	52.35	18	19	0.3233	16.2	0.5425	1423.52502
test1q	1.927	96.35	16	19	0.6515	32.58	1.066	1422.47192
test1s	1.05E-02	52.35	53	19	3.23E-03	16.2	5.43E-03	1426.01257
test1t	1.93E-02	96.35	16	19	6.52E-03	32.58	1.07E-02	1406.3075
test1u	* *							
test1a1	3.33E-16	6.66E-13	10	10	1.19E-16	2.37E-13	8.89E-17	12.665
test1b1	4.23E-16	8.47E-13	3	1	8.56E-17	1.71E-13	1.58E-16	11.55188
test1c1	5.77E-15	1.17E-11	1	1	7.95E-16	1.62E-12	1.11E-15	11.32281
test1d1	3.75E-16	7.49E-13	20	20	1.00E-16	2.00E-13	7.16E-17	12.15469
test1e1	5.93E-09	1.19E-05	42	7	1.26E-09	2.52E-06	2.17E-09	11.94773
test1f1	8.59E-04	4.299	1	2	1.70E-04	0.8491	2.87E-04	11.84161
test1g1	3.75E-16	7.49E-13	1	16	2.20E-16	4.40E-13	9.08E-06	11.85426
test1h1	4.77E-09	9.54E-06	12	10	1.26E-09	2.51E-06	2.87E-07	11.78684
test1i1	8.53E-04	4.273	1	10	2.31E-04	1.159	3.03E-04	11.69456
test1j1	3.68E-16	7.36E-13	48	18	1.59E-16	3.19E-13	9.60E-06	11.6107
test1k1	7.96E-09	1.59E-05	23	18	1.49E-09	2.97E-06	3.04E-07	11.68669
test1l1	8.36E-04	4.21	1	6	2.40E-04	1.209	3.95E-04	11.63261
test1a2	3.02E-14	6.04E-13	10	10	1.07E-14	2.14E-13	8.30E-15	12.755
test1b2	2.80E-14	5.60E-13	10	10	1.32E-14	2.65E-13	7.23E-15	11.56813
test1c2	6.22E-14	3.16E-12	10	10	1.65E-14	8.35E-13	3.59E-14	11.23031
test1d2	3.02E-14	6.04E-13	50	20	1.23E-14	2.46E-13	5.86E-15	12.11453
test1e2	9.44E-07	1.89E-05	23	17	3.86E-07	7.71E-06	6.39E-07	12.04914
test1f2	2.22E-02	1.111	1	11	7.22E-03	0.3609	7.89E-03	11.95164
test1g2	3.11E-14	6.22E-13	50	17	1.86E-14	3.71E-13	2.50E-04	11.88336
test1h2	9.51E-07	1.90E-05	28	4	2.51E-07	5.03E-06	7.91E-06	11.7459
test1i2	2.22E-02	1.111	1	10	7.22E-03	0.3609	1.53E-02	11.67442
test1j2	3.20E-14	6.40E-13	3	13	1.45E-14	2.91E-13	4.84E-04	11.75073
test1k2	6.30E-07	1.26E-05	50	11	2.62E-07	5.25E-06	1.53E-05	11.68654
test1l2	2.23E-02	1.116	50	15	7.47E-03	0.3744	1.24E-02	11.59268
test1m2	1.99E-13	3.98E-13	1	10	7.16E-14	1.43E-13	1.24E-03	10.64949
test1n2	4.89E-15	4.79E-13	10	1	1.57E-15	1.54E-13	1.25E-04	10.65094
test1o2	7.29E-17	3.64E-13	1	9	2.17E-17	1.08E-13	1.26E-05	10.57236
test1p2	0.3901	19.57	70	2	0.1039	5.212	0.1756	1409.67249
test1q2	2.737	137.3	50	2	1.03	51.68	1.708	1410.99902
test1s2	3.55E-02	177.8	26	3	1.21E-02	60.59	1.94E-02	1425.35754
test1t2	2.84E-02	142.4	30	10	1.35E-02	67.69	2.08E-02	1429.50098
test1u2	* *							
test1a12	3.33E-16	6.66E-13	10	10	1.19E-16	2.37E-13	0.1717	9.02808
test1b12	3.13E-16	6.26E-13	10	10	1.27E-16	2.54E-13	1.73E-02	9.06445
test1c12	1.20E-15	2.43E-12	7	10	3.65E-16	7.40E-13	1.73E-03	9.02075
test1d12	3.75E-16	7.49E-13	20	20	1.00E-16	2.00E-13	5.49E-05	10.01733
test1e12	7.13E-09	1.43E-05	26	17	1.36E-09	2.71E-06	1.74E-06	9.93286
test1f12	2.15E-04	1.079	1	2	3.30E-05	0.1655	7.01E-05	9.99878
test1g12	3.75E-16	7.49E-13	1	16	2.20E-16	4.40E-13	2.22E-06	9.99438
test1h12	3.78E-09	7.56E-06	7	16	1.40E-09	2.80E-06	7.02E-08	9.98022
test1i12	2.22E-04	1.111	1	10	7.22E-05	0.3609	1.53E-04	9.94604
test1j12	3.68E-16	7.36E-13	48	18	1.59E-16	3.19E-13	4.84E-06	10.01172
test1k12	4.98E-09	9.96E-06	33	10	1.50E-09	3.00E-06	1.53E-07	9.92749
test1l12	2.23E-04	1.116	50	15	7.47E-05	0.3744	1.24E-04	9.89331
test1aa	5.00E-07	9.69E-06	5	4	1.80E-07	3.49E-06	1.81E-07	12.545
test1bb	8.84E-07	1.77E-05	7	4	2.03E-07	4.07E-06	2.05E-07	11.57563
test1cc	6.31E-07	3.08E-05	4	2	1.35E-07	6.60E-06	1.38E-07	11.23469
test1dd	7.21E-07	1.39E-05	19	9	2.24E-07	4.31E-06	2.24E-07	12.07641

Table 11.1 Statistical Summary of Test Set 1 using Infinite-Plate Spline (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1ee	1.14E-06	2.28E-05	35	3	3.32E-07	6.64E-06	3.19E-07	12.05148
test1ff	8.53E-02	4.069	50	11	2.31E-02	1.104	5.92E-02	11.91238
test1gg	7.21E-07	1.39E-05	32	12	2.24E-07	4.31E-06	1.87E-03	11.81469
test1hh	1.05E-06	2.09E-05	12	6	2.21E-07	4.43E-06	5.93E-05	11.77773
test1ii	8.53E-02	4.112	50	10	2.31E-02	1.115	5.92E-02	11.70518
test1jj	6.52E-07	1.26E-05	17	11	2.34E-07	4.54E-06	1.87E-03	11.71156
test1kk	1.17E-06	2.35E-05	29	18	2.62E-07	5.23E-06	5.93E-05	11.66774
test1ll	8.36E-02	4.03	50	6	2.40E-02	1.157	6.06E-02	11.5739
test1mm	4.70E-06	9.37E-06	6	5	1.47E-06	2.94E-06	6.09E-03	10.63062
test1nn	8.29E-07	7.06E-05	5	9	2.35E-07	2.00E-05	6.12E-04	10.58217
test1oo	6.60E-08	3.80E-05	4	5	8.05E-09	4.64E-06	6.15E-05	10.53369
test1pp	1.045	48.48	53	19	0.3233	15.01	0.4395	1433.59375
test1qq	1.927	89.54	16	19	0.6515	30.28	0.6385	1432.24756
test1ss	1.05E-02	5.514	53	19	3.23E-03	1.707	4.02E-03	1430.52588
test1tt	1.93E-02	9.924	16	19	6.52E-03	3.356	6.38E-03	1434.94507
test1uu	* *							
test1bb2	5.11E-14	1.02E-12	3	1	1.10E-14	2.19E-13	2.07E-14	12.575
test1cc2	2.42E-13	1.23E-11	1	1	1.84E-14	9.32E-13	3.77E-14	11.48219
test1dd2	3.02E-14	6.04E-13	50	20	1.23E-14	2.46E-13	5.88E-15	12.24953
test1ee2	1.01E-06	2.01E-05	27	6	2.94E-07	5.88E-06	5.39E-07	12.09461
test1ff2	8.53E-02	4.273	50	11	2.31E-02	1.159	4.50E-02	11.98952
test1gg2	3.11E-14	6.22E-13	50	17	1.86E-14	3.71E-13	1.43E-03	11.92195
test1hh2	9.07E-07	1.81E-05	12	18	2.04E-07	4.07E-06	4.51E-05	11.80414
test1ii2	8.53E-02	4.273	50	10	2.31E-02	1.159	3.03E-02	11.73644
test1jj2	3.20E-14	6.40E-13	3	13	1.45E-14	2.91E-13	9.60E-04	11.62959
test1kk2	9.79E-07	1.96E-05	22	19	2.49E-07	4.98E-06	3.04E-05	11.70583
test1ll2	8.36E-02	4.21	50	6	2.40E-02	1.209	3.95E-02	11.64183
test1mm2	1.99E-13	3.98E-13	1	10	7.16E-14	1.43E-13	3.97E-03	10.65872
test1nn2	5.77E-15	5.66E-13	1	9	1.86E-15	1.82E-13	3.99E-04	10.64015
test1oo2	1.23E-16	6.16E-13	2	1	4.76E-17	2.38E-13	4.01E-05	10.59155
test1pp2	1.045	52.35	18	19	0.3233	16.2	0.5425	1447.79504
test1qq2	1.927	96.35	16	19	0.6515	32.58	1.066	1430.07251
test1ss2	1.05E-02	52.35	53	19	3.23E-03	16.2	2.90E-02	1422.37842
test1tt2	* *							
test1uu2	* *							
test1A	3.91E-14	3.91E-13	1	9	1.37E-14	1.37E-13	9.31E-15	12.675
test1B	4.86E-14	6.98E-13	9	10	1.07E-14	1.54E-13	2.33E-14	11.55844
test1C	1.48E-06	1.48E-05	17	3	3.86E-07	3.86E-06	6.48E-07	12.21531
test1D	1.47E-06	1.47E-05	17	1	3.52E-07	3.52E-06	6.15E-07	12.11227
test1E	8.36E-02	1.197	1	6	2.40E-02	0.3438	3.95E-02	11.87871
test1F	2.23E-02	0.3181	1	15	7.47E-03	0.1067	1.24E-02	11.86148
test1G	1.07E-06	5.26E-05	11	10	3.09E-07	1.52E-05	3.93E-04	11.84387
test1H	2.23E-02	0.7382	50	15	7.47E-03	0.2477	1.24E-02	11.74586
test1I	4.14E-02	1.037	4	18	5.22E-03	0.1307	5.22E-03	1657.54126
test1J	* *							
test1A1	8.33E-07	8.33E-06	6	9	2.37E-07	2.37E-06	2.38E-07	13.105
test1B1	8.29E-07	1.19E-05	9	5	1.75E-07	2.51E-06	1.77E-07	11.855
test1C1	1.71E-06	1.71E-05	11	19	4.07E-07	4.07E-06	4.07E-07	12.97422
test1D1	1.77E-06	1.77E-05	15	9	3.74E-07	3.74E-06	3.74E-07	12.60062
test1E1	8.36E-02	1.197	1	6	2.40E-02	0.3437	2.40E-02	12.33062
test1F1	2.23E-02	0.3181	1	15	7.47E-03	0.1067	7.51E-03	12.37172
test1G1	1.21E-06	5.55E-05	26	3	3.09E-07	1.41E-05	2.38E-04	11.95195
test1H1	2.23E-02	0.7158	50	15	7.47E-03	0.2402	7.47E-03	11.83327
test1I1	4.14E-02	1.016	4	18	5.22E-03	0.128	5.22E-03	1695.95618
test1J2	* *							
test1A2	1.10E-06	1.10E-05	1	2	3.82E-07	3.82E-06	7.51E-04	10.83333
test1B2	1.10E-06	1.54E-05	2	1	2.91E-07	4.07E-06	7.55E-05	10.77463
test1C2	5.20E-06	5.10E-05	3	7	6.21E-07	6.09E-06	2.46E-06	11.69545
test1D2	1.52E-06	1.52E-05	3	7	3.90E-07	3.90E-06	3.98E-07	11.59158
test1E2	8.36E-02	1.172	1	6	2.40E-02	0.3365	2.40E-02	11.68784
test1F2	2.23E-02	0.3116	1	15	7.47E-03	0.1046	7.51E-03	11.48354
test1G2	1.02E-06	4.22E-05	35	13	3.11E-07	1.28E-05	2.38E-04	11.57962
test1H2	2.23E-02	0.6631	50	15	7.47E-03	0.2225	7.47E-03	11.87543

Table 11.2 Statistical Summary of Test Set 2 using Infinite-Plate Spline

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test2A	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	1.08
test2B	4.00E-07	2.00E-05	10	6	2.00E-07	1.00E-05	3.18E-07	1.4275
test2C	4.00E-07	2.00E-05	5	10	2.00E-07	1.00E-05	3.19E-07	1.57125
test2D	8.00E-07	1.33E-05	3	5	2.45E-07	4.09E-06	4.37E-07	1.6275
test2E	0.5384	27.34	8	7	0.1517	7.704	0.2802	1.71375
test2F	1.02E-14	5.11E-13	28	50	3.68E-15	1.84E-13	5.60E-03	0.54938
test2G	5.24E-07	2.62E-05	14	3	2.64E-07	1.32E-05	1.12E-04	0.88594
test2H	5.24E-07	2.62E-05	3	43	2.64E-07	1.32E-05	2.28E-06	1.04266
test2I	9.03E-07	1.51E-05	46	19	2.52E-07	4.20E-06	4.22E-07	1.19312
test2J	* * *							
test2K	* * *							
test2L	1.30E-09	1.30E-05	17	21	2.47E-10	2.47E-06	8.46E-09	1.23984
test2M	1.37E-09	1.37E-05	17	32	2.34E-10	2.34E-06	4.18E-10	1.30969
test2N	* * *							
test2A1	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	1.08
test2B1	5.49E-07	2.75E-05	7	8	1.93E-07	9.64E-06	1.95E-07	1.4275
test2C1	5.49E-07	2.75E-05	8	7	1.93E-07	9.64E-06	1.95E-07	1.555
test2D1	9.75E-07	1.62E-05	3	3	2.32E-07	3.86E-06	2.34E-07	1.61
test2E1	0.5384	26.8	8	7	0.1517	7.553	0.1525	1.72
test2F1	6.06E-07	2.78E-05	10	44	2.36E-07	1.09E-05	3.05E-03	0.5325
test2G1	6.06E-07	3.03E-05	23	16	2.39E-07	1.20E-05	6.10E-05	0.89094
test2H1	6.04E-07	3.02E-05	16	23	2.39E-07	1.20E-05	1.24E-06	1.04953
test2I1	1.02E-06	1.70E-05	48	18	2.52E-07	4.19E-06	2.53E-07	1.16297
test2J1	* * *							
test2K1	* * *							
test2L1	9.74E-08	5.30E-05	25	28	1.13E-09	6.16E-07	5.18E-09	1.29031
test2M1	9.73E-08	5.30E-05	28	26	1.15E-09	6.28E-07	1.16E-09	1.29367
test2N1	* * *							

Table 11.3 Statistical Summary of Test Set 3 using Infinite-Plate Splines.

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A	1.04E-06	2.09E-05	4	5	3.10E-07	6.21E-06	4.92E-07	12.025
test3B	0.3119	8.949	1	1	3.62E-02	1.038	0.109	10.82875
test3C	9.93E-07	1.99E-05	2	1	3.17E-07	6.35E-06	3.45E-03	11.66859
test3D	1.17E-06	2.35E-05	9	4	2.97E-07	5.93E-06	1.09E-04	11.43976
test3E	0.9325	26.7	1	8	0.2989	8.556	0.666	11.36961
test3F	0.3119	8.918	1	1	8.41E-02	2.404	0.1989	11.44456
test3G	1.18E-06	1.15E-04	3	11	3.10E-07	3.04E-05	6.29E-03	11.24863
test3H	0.3119	20.69	1	1	8.41E-02	5.578	0.1977	11.30336
test3I	* * *							
test3J	* * *							
test3A1	9.90E-07	1.98E-05	1	8	2.94E-07	5.88E-06	2.96E-07	11.835
test3B1	0.3119	8.949	1	1	3.63E-02	1.041	0.3146	10.79656
test3C1	1.25E-06	2.50E-05	18	3	3.42E-07	6.85E-06	9.95E-03	11.49265
test3D1	1.30E-06	2.59E-05	7	2	2.94E-07	5.88E-06	3.15E-04	11.41195
test3E1	0.9325	26.7	1	8	0.2998	8.582	0.3986	11.27337
test3F1	0.3119	8.918	1	1	8.41E-02	2.404	0.3673	11.12695
test3G1	1.19E-06	1.01E-04	8	10	3.23E-07	2.75E-05	1.16E-02	11.04184
test3H1	0.3119	19.45	1	1	8.41E-02	5.241	8.41E-02	11.01715
test3I1	* * *							
test3J1	* * *							
test3A2	9.90E-07	1.98E-05	1	8	3.06E-07	6.11E-06	8.45E-03	10.00286
test3B2	0.3119	8.949	1	1	3.63E-02	1.041	0.3146	9.98444
test3C2	1.25E-06	2.50E-05	18	3	3.42E-07	6.85E-06	9.95E-03	10.9368
test3D2	1.30E-06	2.59E-05	7	2	2.94E-07	5.88E-06	3.15E-04	10.87434
test3E2	0.9325	26.7	1	8	0.2998	8.582	0.3986	10.87207
test3F2	0.3119	8.918	1	1	8.41E-02	2.404	0.3673	10.79977
test3G2	1.19E-06	1.01E-04	8	10	3.23E-07	2.75E-05	1.16E-02	10.77734
test3H2	0.3119	19.45	1	1	8.41E-02	5.241	8.41E-02	10.75516
test3I2	* * *							
test3J2	* * *							
test3A3	1.27E-14	6.33E-13	1	10	3.56E-15	1.78E-13	1.99E-02	3.13723
test3B3	4.00E-07	2.00E-05	10	5	2.00E-07	1.00E-05	2.00E-03	3.12957
test3C3	4.00E-07	2.00E-05	5	4	2.00E-07	1.00E-05	2.01E-04	3.14104
test3D3	8.00E-07	1.33E-05	3	5	2.47E-07	4.11E-06	2.02E-05	3.14233
test3E3	0.7078	35.94	8	1	0.197	10	0.3254	3.15359
test3F3	1.29E-14	6.44E-13	10	42	5.08E-15	2.54E-13	6.51E-03	1.80644
test3G3	5.24E-07	2.62E-05	47	3	2.64E-07	1.32E-05	1.30E-04	1.79959
test3H3	5.23E-07	2.62E-05	3	24	2.64E-07	1.32E-05	2.63E-06	1.82222
test3I3	9.01E-07	1.50E-05	46	19	2.51E-07	4.18E-06	4.13E-07	0.17875
test3J3	* * *							
test3A4	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	8.45E-03	3.37643
test3B4	5.49E-07	2.75E-05	7	8	1.93E-07	9.64E-06	8.50E-04	3.38771
test3C4	5.49E-07	2.75E-05	8	7	1.93E-07	9.65E-06	8.54E-05	3.37907
test3D4	9.75E-07	1.63E-05	3	3	2.34E-07	3.91E-06	8.58E-06	3.38016
test3E4	0.7078	35.23	8	1	0.2001	9.959	0.5143	3.38077
test3F4	6.06E-07	2.78E-05	10	44	2.36E-07	1.09E-05	1.03E-02	2.03232
test3G4	6.06E-07	3.03E-05	23	16	2.39E-07	1.20E-05	2.06E-04	2.01874
test3H4	6.04E-07	3.02E-05	16	23	2.39E-07	1.20E-05	4.12E-06	2.04532
test3I4	* * *							
test3J4	* * *							

Table 11.4 Statistical Summary of Test Set 4 using Infinite-Plate Splines.

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test4A2	1.14E-06	2.27E-05	22	11	4.27E-07	8.55E-06	6.49E-07	0.51
test4A3	1.43E-06	2.87E-05	31	28	4.38E-07	8.76E-06	7.01E-07	2.935
test4B2	6.07E-02	1.747	36	12	2.40E-02	0.6902	3.90E-02	0.39781
test4B3	6.12E-02	1.76	73	23	2.43E-02	0.6986	3.92E-02	2.51094
test4C2	7.84E-07	1.57E-05	129	75	4.60E-07	9.20E-06	3.10E-04	8.36922
test4C3	1.05E-06	2.10E-05	317	42	5.09E-07	1.02E-05	1.35E-06	38.31399
test4D2	1.32E-06	2.64E-05	153	6	4.70E-07	9.39E-06	6.31E-07	7.85512
test4D3	1.40E-06	2.79E-05	54	70	4.72E-07	9.43E-06	6.30E-07	39.50633
test4E2	8.80E-02	2.529	188	61	2.55E-02	0.7331	5.05E-02	7.75046
test4E3	8.81E-02	2.53	376	38	2.56E-02	0.7351	5.07E-02	37.99612
test4F2	5.91E-02	1.698	110	46	1.19E-02	0.3429	2.37E-02	7.51468
test4F3	6.10E-02	1.755	220	69	1.21E-02	0.3476	2.39E-02	37.6907
test4G2	1.03E-06	1.01E-04	160	48	4.73E-07	4.64E-05	1.89E-04	7.34967
test4G3	1.04E-06	1.02E-04	237	13	4.84E-07	4.75E-05	1.03E-06	37.58569
test4H2	5.91E-02	3.964	110	46	1.19E-02	0.8003	2.37E-02	7.36456
test4H3	6.10E-02	4.104	221	69	1.21E-02	0.813	2.39E-02	37.81006
test4A12	0.1899	3.797	20	20	4.63E-02	0.9252	4.72E-02	0.57
test4A13	0.195	3.899	40	41	4.91E-02	0.9828	4.69E-02	2.91188
test4B12	0.1914	5.508	21	21	5.31E-02	1.528	5.48E-02	0.37625
test4B13	0.1958	5.628	40	41	5.45E-02	1.566	5.42E-02	2.49281
test4C12	0.183	3.66	176	40	1.09E-02	0.2174	2.41E-02	8.35938
test4C13	0.1893	3.786	351	81	1.08E-02	0.2159	2.38E-02	38.35422
test4D12	0.19	3.8	174	41	1.14E-02	0.2288	2.72E-02	7.9252
test4D13	0.192	3.84	350	81	1.09E-02	0.2171	2.41E-02	37.92737
test4E12	0.1898	5.455	174	40	2.82E-02	0.8111	5.14E-02	7.68256
test4E13	0.1947	5.59	349	80	2.84E-02	0.8141	5.17E-02	38.04831
test4F12	0.1943	5.579	174	43	1.71E-02	0.4918	3.19E-02	7.54669
test4F13	0.1959	5.636	349	81	1.73E-02	0.4984	3.21E-02	37.98264
test4G12	0.1883	15.63	175	41	9.60E-03	0.7962	2.18E-02	7.84113
test4G13	0.1843	15.24	344	81	1.07E-02	0.8845	2.36E-02	37.8858
test4H12	0.1943	12.31	174	43	1.71E-02	1.085	3.19E-02	7.4541
test4H13	0.1959	12.46	349	81	1.73E-02	1.102	3.21E-02	37.9292
test4A22	0.2848	5.696	20	21	7.96E-02	1.592	7.46E-02	0.5875
test4A23	0.2925	5.849	41	41	7.44E-02	1.487	7.00E-02	2.96625
test4B22	0.2864	8.24	20	21	7.53E-02	2.167	7.46E-02	0.38906
test4B23	0.2933	8.43	41	41	7.71E-02	2.215	7.33E-02	2.47922
test4C22	0.183	3.66	176	40	1.09E-02	0.2174	2.41E-02	8.43797
test4C23	0.1893	3.786	351	81	1.08E-02	0.2159	2.38E-02	38.24308
test4D22	0.19	3.8	174	41	1.14E-02	0.2288	2.72E-02	7.78474
test4D23	0.192	3.84	350	81	1.09E-02	0.2171	2.41E-02	37.91718
test4E22	0.1898	5.455	174	40	2.82E-02	0.8111	5.14E-02	7.68256
test4E23	0.1947	5.59	349	80	2.84E-02	0.8141	5.17E-02	38.00839
test4F22	0.1943	5.579	174	43	1.71E-02	0.4918	3.19E-02	7.55695
test4F23	0.1959	5.636	349	81	1.73E-02	0.4984	3.21E-02	37.55292
test4G22	0.1883	15.63	175	41	9.60E-03	0.7962	2.18E-02	7.38202
test4G23	0.1843	15.24	344	81	1.07E-02	0.8845	2.36E-02	37.49805
test4H22	0.1943	12.31	174	43	1.71E-02	1.085	3.19E-02	7.38702
test4H23	0.1959	12.46	349	81	1.73E-02	1.102	3.21E-02	37.64252

Table 11.5 Statistical Summary of Test Set 1 (Partial) using Single Precision Infinite-Plate Splines.

Test Case	Maximum Error				Average Error		Standard
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation
test1a	1.29E-05	2.58E-04	10	7	4.26E-06	8.52E-05	7.92E-06
test1b	1.76E-05	3.53E-04	2	1	5.09E-06	1.02E-04	1.07E-05
test1c	2.71E-04	1.38E-02	3	1	6.20E-05	3.15E-03	4.44E-05
test1d	1.05E-05	2.10E-04	41	1	5.51E-06	1.10E-04	1.13E-05
test1e	2.00E-05	4.01E-04	29	1	1.12E-05	2.23E-04	2.28E-05
test1f	8.53E-02	4.277	50	11	2.31E-02	1.157	4.49E-02
test1g	1.14E-05	2.29E-04	25	4	3.50E-06	7.00E-05	1.42E-03
test1h	1.76E-05	3.53E-04	2	1	3.00E-06	5.99E-05	4.54E-05
test1i	8.54E-02	4.281	50	10	2.32E-02	1.161	3.03E-02
test1j	1.05E-05	2.10E-04	23	1	3.87E-06	7.75E-05	9.59E-04
test1k	1.86E-05	3.72E-04	14	2	6.55E-06	1.31E-04	3.34E-05
test1l	8.38E-02	4.22	50	15	2.40E-02	1.209	3.94E-02
test1m	1.30E-04	2.59E-04	10	7	4.83E-05	9.65E-05	3.96E-03
test1n	3.82E-06	3.74E-04	4	1	1.18E-06	1.16E-04	3.98E-04
test1o	5.40E-08	2.70E-04	8	10	1.56E-08	7.81E-05	4.00E-05
test1p	1.05	52.6	18	19	0.3238	16.23	0.5432
test1q	1.929	96.47	85	19	0.6517	32.59	1.065
test1a1	1.30E-07	2.61E-04	4	1	4.93E-08	9.86E-05	9.44E-08
test1b1	1.57E-07	3.13E-04	2	1	4.06E-08	8.12E-05	8.31E-08
test1c1	6.00E-06	1.22E-02	3	1	1.29E-06	2.63E-03	1.17E-06
test1d1	1.38E-07	2.76E-04	41	3	7.48E-08	1.50E-04	1.57E-07
test1e1	1.68E-07	3.35E-04	29	1	9.53E-08	1.91E-04	1.95E-07
test1f1	8.61E-04	4.304	50	2	1.70E-04	0.8491	2.87E-04
test1g1	1.34E-07	2.68E-04	5	1	5.38E-08	1.08E-04	9.07E-06
test1h1	1.57E-07	3.13E-04	2	1	2.72E-08	5.45E-05	2.89E-07
test1i1	8.53E-04	4.277	50	10	2.32E-04	1.16	3.03E-04
test1j1	1.34E-07	2.68E-04	23	1	5.80E-08	1.16E-04	9.60E-06
test1k1	1.64E-07	3.28E-04	17	2	5.16E-08	1.03E-04	3.21E-07
test1l1	8.37E-04	4.216	50	15	2.40E-04	1.209	3.95E-04
test1a2	1.29E-05	2.58E-04	10	7	4.26E-06	8.52E-05	7.92E-06
test1b2	9.59E-06	1.92E-04	8	10	2.85E-06	5.70E-05	5.37E-06
test1c2	1.72E-05	8.72E-04	3	1	2.68E-06	1.36E-04	5.66E-06
test1d2	1.05E-05	2.10E-04	41	1	5.51E-06	1.10E-04	1.12E-05
test1e2	9.72E-06	1.95E-04	48	20	1.80E-06	3.60E-05	3.36E-06
test1f2	2.22E-02	1.111	50	11	7.21E-03	0.3607	7.89E-03
test1g2	1.14E-05	2.29E-04	25	4	3.50E-06	7.00E-05	2.50E-04
test1h2	1.01E-05	2.01E-04	15	16	5.40E-06	1.08E-04	1.35E-05
test1i2	2.22E-02	1.112	1	10	7.22E-03	0.361	1.53E-02
test1j2	1.05E-05	2.10E-04	23	1	3.87E-06	7.75E-05	4.84E-04
test1k2	1.02E-05	2.03E-04	32	19	3.12E-06	6.24E-05	1.66E-05
test1l2	2.23E-02	1.116	1	15	7.47E-03	0.3743	1.24E-02
test1m2	1.30E-04	2.59E-04	10	7	4.83E-05	9.65E-05	1.25E-03
test1n2	2.86E-06	2.81E-04	10	7	8.98E-07	8.80E-05	1.25E-04
test1a12	1.30E-07	2.61E-04	4	1	4.93E-08	9.86E-05	1.26E-05
test1b12	1.11E-07	2.21E-04	8	10	3.65E-08	7.31E-05	1.27E-06
test1c12	4.17E-07	8.47E-04	3	1	5.97E-08	1.21E-04	1.68E-07
test1d12	1.38E-07	2.76E-04	41	3	7.48E-08	1.50E-04	1.53E-07
test1e12	1.11E-07	2.22E-04	48	20	2.87E-08	5.74E-05	3.08E-08
test1aa	1.29E-05	2.50E-04	10	7	4.26E-06	8.27E-05	3.81E-06
test1bb	1.76E-05	3.53E-04	2	1	5.21E-06	1.04E-04	8.70E-06
test1cc	2.71E-04	1.32E-02	3	1	6.19E-05	3.02E-03	6.79E-05
test1dd	1.05E-05	2.02E-04	41	1	5.51E-06	1.06E-04	5.47E-06
test1ee	2.00E-05	4.01E-04	29	1	1.12E-05	2.24E-04	1.65E-05
test1ff	8.53E-02	4.074	50	11	2.31E-02	1.102	5.90E-02
test1gg	1.14E-05	2.20E-04	24	4	3.51E-06	6.75E-05	1.87E-03
test1hh	1.76E-05	3.53E-04	2	1	2.99E-06	5.98E-05	5.93E-05
test1ii	8.54E-02	4.119	50	10	2.32E-02	1.117	5.92E-02

11.2 Biharmonic-Multiquadrics Method

MQ is a method for true scattered data, a scheme for representing surfaces and bodies in an arbitrary dimensional space. As described before, monotonicity and convexity are observed properties of the method, and it is not only used for interpolation but for estimation of partial derivatives.

The implementation of MQ is described in Chapter 5. Similar to the Thin-Plate Spline method, some options were explored. Among them: the issue of scaling the data to a unit domain or using the data as given; a global implementation versus local (i.e., domain decomposition or subdomaining); and the presence of the *r* parameter to be set by the user or be internally evaluated. These options were investigated using a partial set of test set 1 to determine the most efficient, yet accurate method of implementing MQ.

<u>Overall Accuracy</u> – Statistical summaries of the test cases are presented in Tables 11.6 – 11.10. The overall accuracy of the MQ method is very good. Even though the error varies across the range of test case functions, it is usually less than 10% of the function amplitude or magnitude. The largest errors occurred in the sinusoidal cases with higher number of cycles. Figures 11-11 and 11-12 plot the error for each category of function test cases. It is directly observed from these plots that the method has excellent performance for constant and most linear and sinusoidal functions. There is, however, a tendency towards larger errors for the higher-frequency sinusoidal functions, even though the relative error remains below 5%. For most runs, the maximum and the average errors are of the same order of magnitude, indicating that the method is not sensitive to any particular location on the surface.

<u>Grid Spacing Sensitivity</u> – The large variation of errors in Figures 11-11 and 11-12 indicates that MQ is sensitive to grid spacing. The constant functions in Figure 11-11 show the interpolation to be insensitive to the grid spacing. However, the linear and sinusoidal function interpolations show a 12 order of magnitude difference in error. The runs marked "A" in Figure 11-11 indicate that the function was interpolated to an identical grid. For the sets of data marked "B", the function was interpolated to a grid which was clustered in near the edges – as found in CFD grids. However, the overall maximum error is less than 10%, with most errors remaining below 1% with respect to the function amplitude or magnitude.

As an example of a characteristic oscillation in the interpolation result, Figure 11-13 shows a typical interpolation error plot using MQ. Figures 11-13 a) and 11-13 b) show the reference and the interpolated one-cycle sinusoidal function, respectively. The errors are primarily concentrated on the border of the domain (Figures 11-13 c) and 11-13 d)).

<u>Directional Bias</u> – Figure 11-11 includes the functions examined in test set 1. These data are virtually identical for both directions, indicating little or no sensitivity to direction of the function.

<u>Magnitude/Amplitude Sensitivity</u> – MQ appears to have no sensitivity to different function magnitudes and/or amplitudes, as shown by the constant errors for different amplitudes in Figure 11-14 for test set 1. This trend is consistent for function type, plates, shells, direction of the function and grid (test sets 1 and 2). This trend is as anticipated since MQ is a linear method.

<u>Sensitivity to Frequency (Higher Oscillations)</u> – MQ is sensitive to the frequency of the function. From the results in Tables 11.6 – 11.9, the errors are very small for constant and linearly-varying functions. Figures 11-11 and 11-12 confirm this trend. There is a very large jump in error when a sinusoidal varying function is introduced. The sinusoidal functions shown in Figures 11-11 and

11-12 are for one cycle over the surface. When the frequency of the sinusoidal function is increased, the error from MQ interpolations increases rapidly. Figure 11-15 shows an increase of 6 – 11 orders of magnitude when the function frequency is increased from one to three cycles. The error remains approximately constant above three cycles since the relative error has reached the same order of magnitude of the function.

The reason for this large jump in error from Figure 11-15 is seen in Figure 11-16. Here, the interpolation result for a three-cycle sinusoid is plotted. Notice that while the function amplitude is captured along the surface edges, the amplitude prediction has slightly degraded along the interior of the grid, as seen by a slight concavity in Figure 11-16 b). The oscillations discussed earlier, and plotted for a one-cycle sinusoid oscillation in Figure 11-13 are now more apparent. These oscillations occur in the direction where the function is supposed to be constant, indicating a directional coupling of the interpolation.

<u>Extrapolation</u> – The maximum relative errors for test sets 1 and 3 are plotted in Figure 11-17 to illustrate the impact of extrapolation on the accuracy of MQ. It is apparent that MQ is not as accurate for extrapolations since the errors increase by approximately four orders of magnitude for linearly varying as well as bi-linearly varying functions. For the test cases with sinusoidal variations, the errors for interpolation and extrapolation are similar (see Figure 11-17). MQ does appear to have problems with the sinusoidal functions in multiple directions. For the larger error results, it is apparent from Table 11.8 that the maximum errors are not due to extrapolation, but to the interior curve fitting. Since most pressures and mode shapes will rapidly vary in the regions which are most likely to require extrapolation (leading edge, trailing edge, root, tip, etc.), the results indicate that special attention should be taken when using the MQ method for extrapolation of high-frequency functions since they can lead to unacceptably large percentage errors.

<u>Diminishing Variation</u> – Figure 11-18 shows that the relative error does not increase as the grids are doubled, indicating that MQ is consistent. If the errors show a change, this would indicate that oscillations are being introduced into the interpolation as the grid becomes finer.

<u>Sensitivity to Three-Dimensional Surfaces (Plates Vs. Shells)</u> – From Figures 11-11 – 11-18, it is seen that the only sensitivity that MQ has when extended from two- to three-dimensional surfaces is in the constant and linear regions. The sensitivities due to sinusoidal functions do not change to any significant degree.

<u>Sensitivity to Grid Irregularities</u> – For the irregular grid test set (test set 2), MQ performed very well for constant and linearly varying functions. There is a large jump in the error when MQ is applied to sinusoidal functions (see Table 11.7). Franke [11] has indicated that the number of elements that define the original grid can be a factor in the accuracy of the method. Since the grid here is very sparse, it seems likely that this causes the errors associated with those rapidly varying functions to increase even more. It is important, therefore, that this method be used with grids that have enough points to adequately identify the shape of the function since MQ does not offer the smoothing that is inherent in some spline methods.

<u>Sensitivity to Subdomaining and Overlapping</u> – For global versus local implementation, some definitions are in order. The term global implies that the entire surface is solved at once, resulting in a single linear system problem. Thus, all points will have a bearing on the interpolation. The term local means that the surface is subdivided into a prescribed number of subdomains. The points within the subdomain are influenced only by the other points within the subdomain. There are some common overlapping regions where the quantities are blended using weighted averages. This option has a bearing on the size of the linear system to be solved.

The association between the size of the linear system and the performance of the MQ scheme was considered. For example, to run a case with an input grid of 150×150, the linear system to be solved would have a $22{,}500 \times 22{,}500$ fully-populated coefficient matrix if the method is used in its global form. In addition to the huge memory requirements to run realistic cases, Kansa [12, 13] points out that the condition number of the coefficient matrix tends to increase with size. He recommends implementing the method with subdomains, which lowers the demand for computer memory by reducing the size of the linear system to be solved. The main advantages of subdomaining are decrease in the overall CPU time requirement, and the reduction of the arrays dimensions, thus reducing the overall memory requirements. In addition, rapidly varying functions can be more accurately interpolated by limiting the zone of influence. A disadvantage of the subdomain approach is that two additional parameters in the form of the subdomain divisions and overlap regions are introduced. One must also ensure that there are an adequate number of points in each subdomain.

The subdomain concept was implemented by allowing a maximum number of input points in each direction (x, y, and z) for a given region. This approximately defines the size of the local linear system to be solved. More points enter in the region through the overlapping areas (which was varied between 10% – 30% of the dimension of the subdomain in each direction). Most cases were run using 20 as the maximum number of points in each direction and 10% overlapping. This gives a reasonable size of sub-problems to be solved, and samples a good portion of the original problem. Reducing the number of points would increase the performance of the code, but it can reduce the accuracy of the interpolation if an insufficient number of points is taken in one or more subdomains.

In order to illustrate the change in the accuracy of the method as function of the subdomaining and scaling, test 1p was used with a fixed 10% overlapping in each direction. Since its input grid is a 50×10, subdivisions were made only along the x-direction, where more points were available. Figure 11-19 shows the error of the solution of test 1p as function of the number of subdomains. Two sets of data were also included to show the influence of scaling each subdomain to a unit square. There is no significant difference in the interpolation results as a function of these two parameters.

<u>Significance of the Additional Parameter r</u>– The parameter r has been introduced in the formulation to make the basis function infinitely differentiable (see Chapter 5). Increasing its value better represents constant derivatives (Franke [11]). As pointed out by Kansa [12, 13], varying r also improves the condition number of the linear system. For all of the test cases in this report, the parameter was varied exponentially from 10^{-5} to 10^{-3}. In addition, no difference in the interpolation results was found when $r < 10^{-5}$ or $r > 1$. Thus, as long as derivatives are not required, the r parameter should be used in the range discussed above. The linear system becomes ill-conditioned, even for some of the small test cases when considering partial derivatives. It appears that r must then be evaluated on a case-by-case basis.

<u>Sensitivity to Planar Curves (Beam-Like Cases)</u> – Tables 11.10 and 11.11 summarize the statistical results obtained from MQ when interpolating data along a planar curve for both scaled and unscaled data. The method performed very well and the maximum overall error was < 1% (<< 1% for most of the cases). For the set of test cases studied, there were no significant differences in scaling achieved by scaling the data.

<u>Algorithm CPU Memory and Time Requirements</u> – The average CPU time requirement is approximately 1–3 seconds (refer to the last column of Tables 11.6 – 11.11). An exception to this

average is for test cases with very large errors, indicative of ill-conditioned matrices. These runs required on the average of 200 to 400 seconds. All these results were computed for a maximum of 20 input data point in each direction of the subdomain and a 10% overlapping of the subregions.

The algorithm CPU memory requirement can be determined using following algorithm:

$$\text{CPU SIZE (megabytes)} = 0.3 + 9.1374\times10^{-4}\,\text{KGS} + 1.0839\times10^{-7}\,\text{KGS}^2 + 3.0867\times10^{-4}\,\text{UKS} + 8.0995\times10^{-11}\,\text{UKS}^2$$

where KGS is the total number of points for the grid where the function is known and UKS is the total number of mesh points for the grid to which the function is to be interpolated. For example, if mode shapes are to be interpolated from a structural mesh to a CFD mesh, KGS is the number of mesh points for the structural mesh, and UKS is the number of surface CFD grid points. The CPU SIZE function is plotted in Figure 11-20. This function yields values within 1% of the actual CPU size during tests.

The influence of subdomaining and the size of the overlapping on the memory requirement is shown in Figure 11-21. The data are based on 150 × 150 input grid and 200 × 200 output grid. There is a drastic reduction in the amount of memory required when the subdomaining technique is used, reflecting directly on the CPU time required. Figure 11-22 shows the actual memory requirements and CPU time for the test 1p, previously used to study the accuracy of the subdomaining.

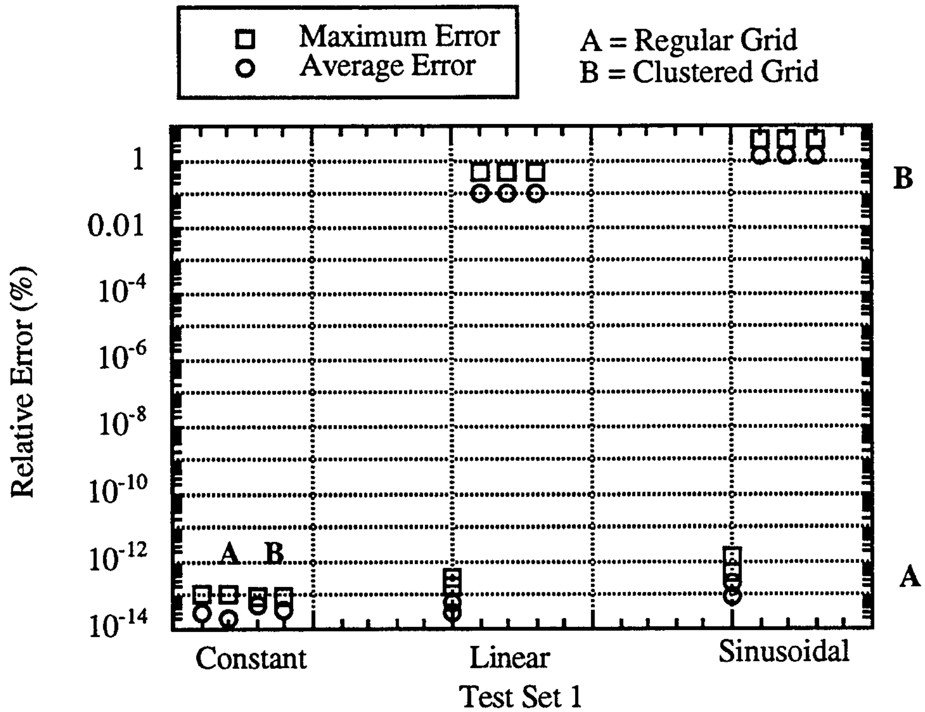


Figure 11-11. Variation of Error for Test Set One Based Upon Function Type for Plates Using Multiquadrics

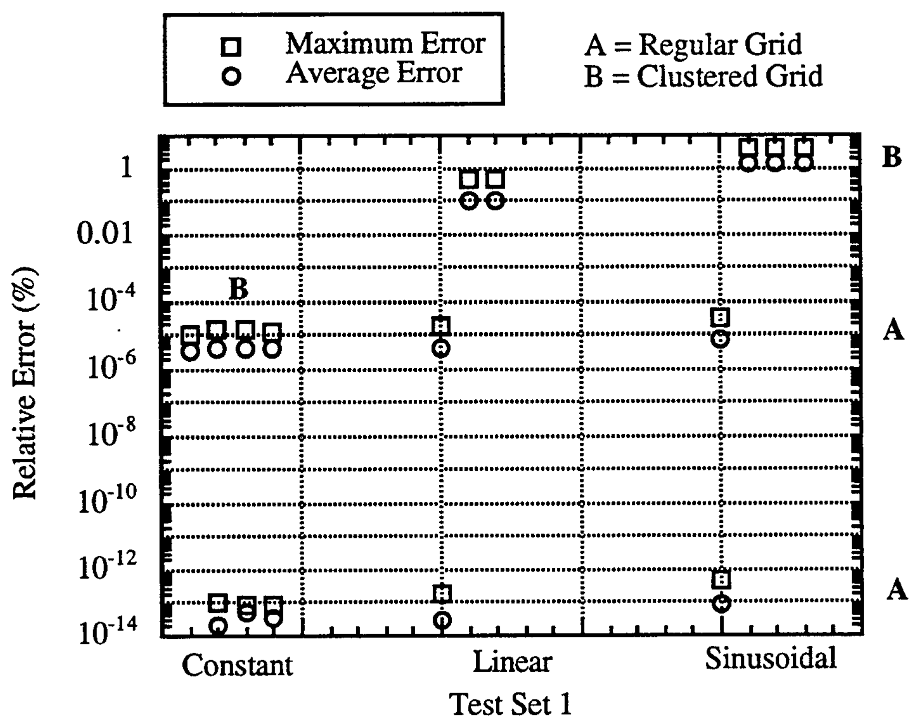
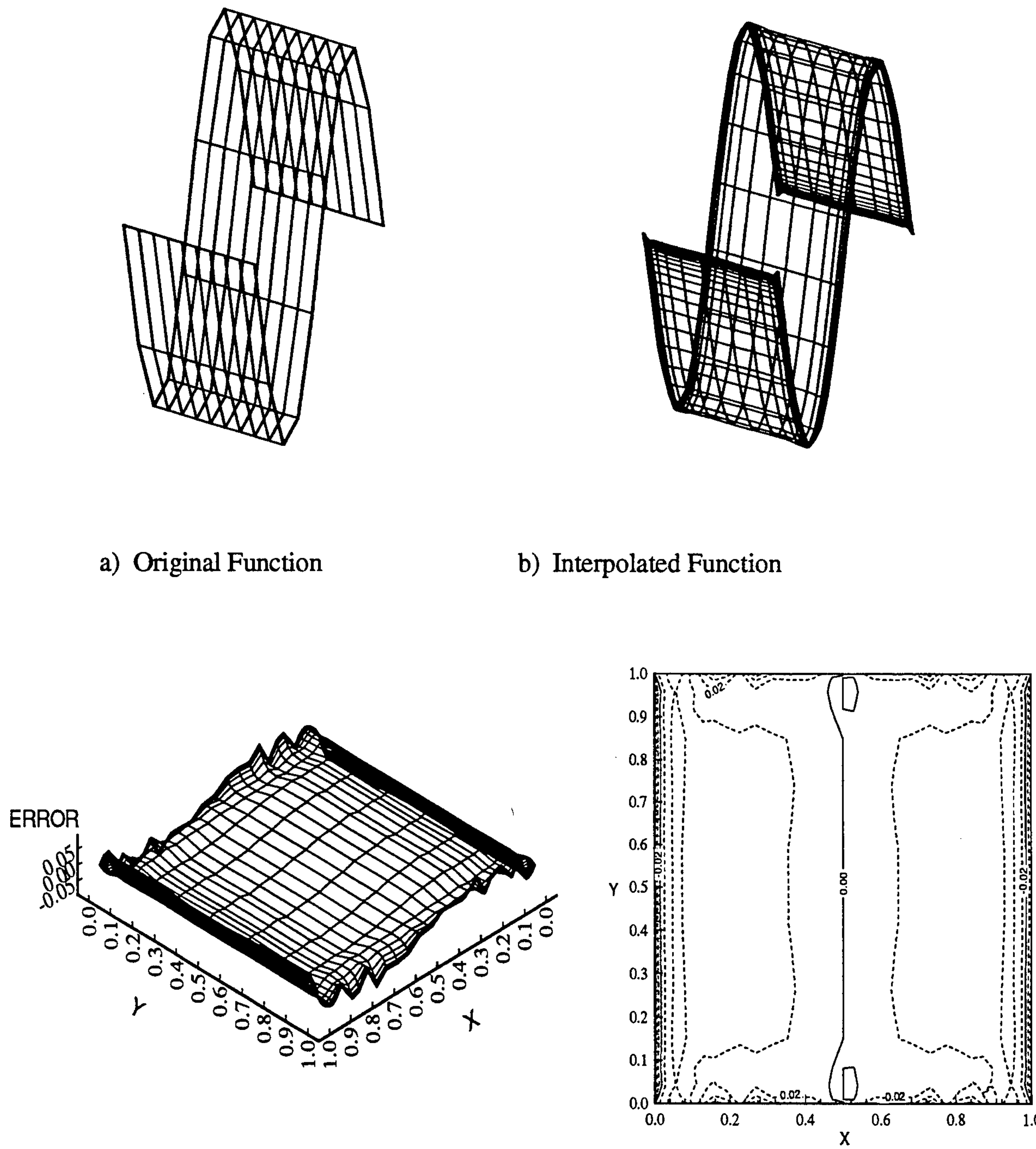


Figure 11-12. Variation of Error for Test Set One Based Upon Function Type for Shells Using Multiquadrics



a) Original Function b) Interpolated Function

c) Orthogonal View of the Error d) Error Contours

Figure 11-13. Example of Oscillations Induced by the Multiquadrics Method (Test 11) for a One-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2

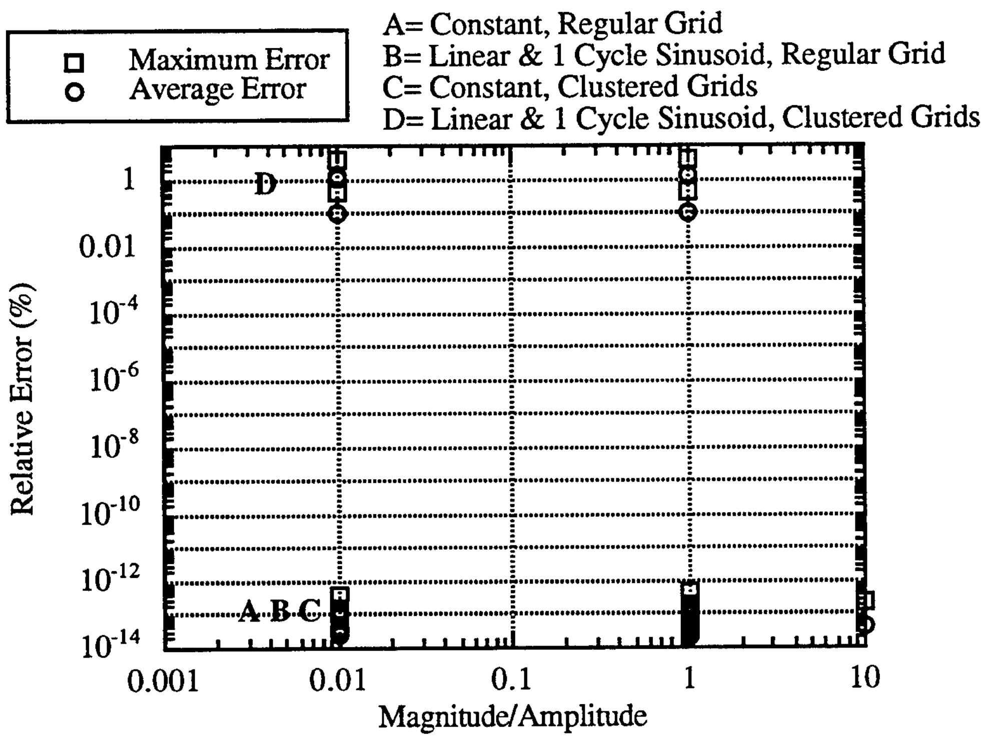


Figure 11-14. Variation of Error for Test Set One Based Upon the Order of Magnitude of the Function

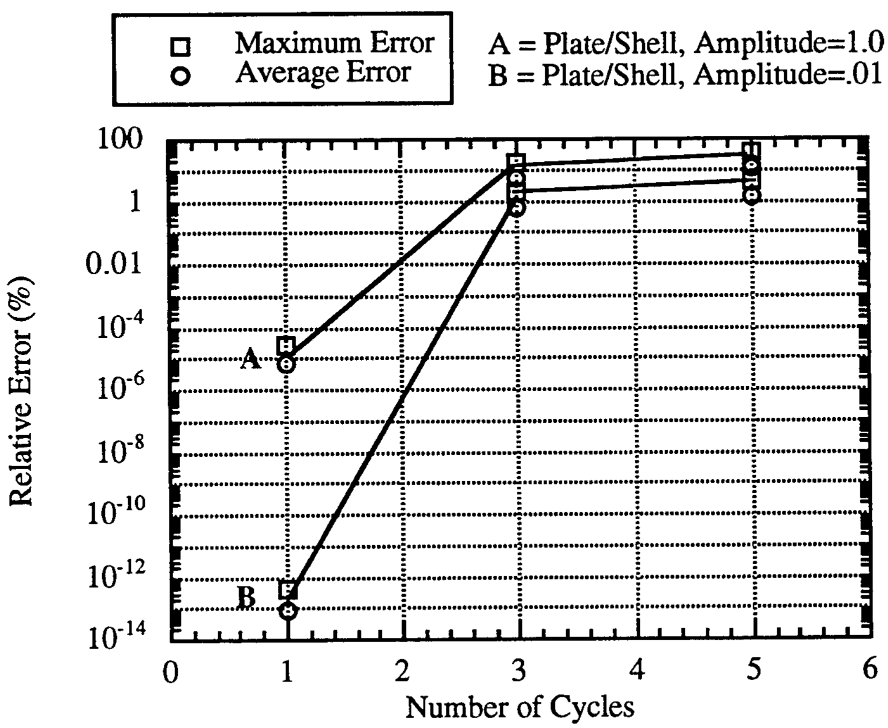
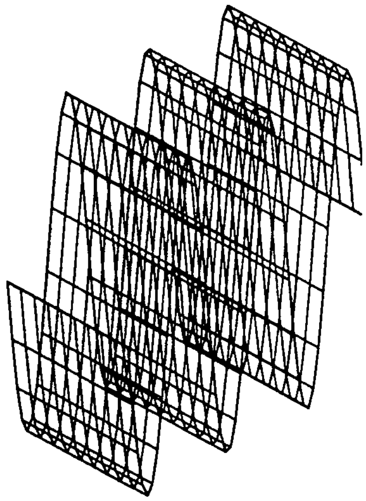
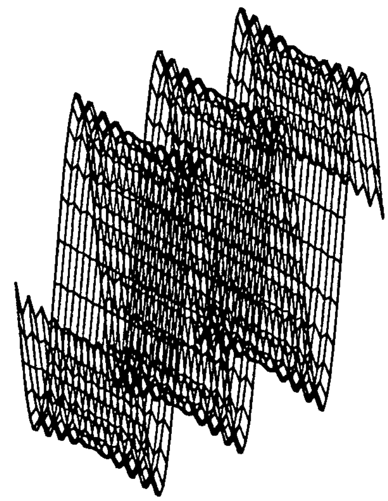


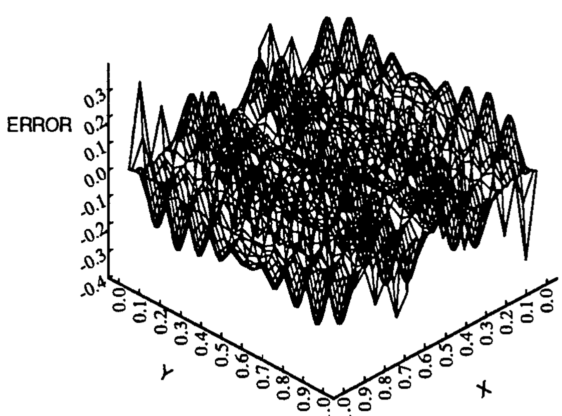
Figure 11-15. Variation of Error for Test Set One Based Upon the Number of Sinusoidal Cycles on the Surface Using Multiquadrics



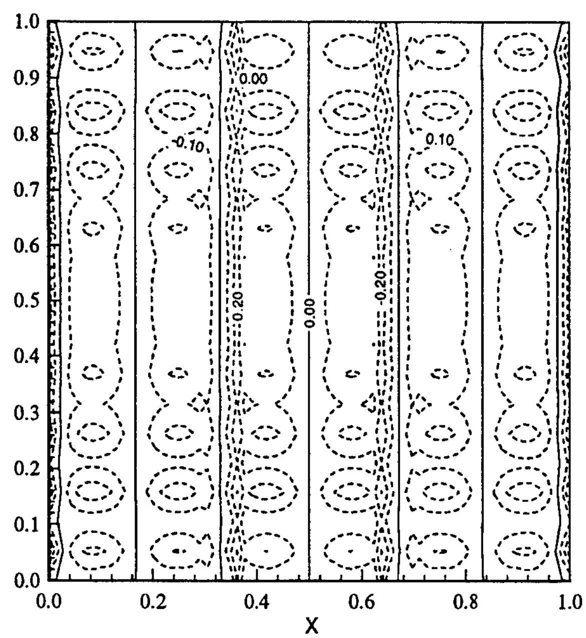
a) Original Function



b) Interpolated Function



c) Orthogonal View of Error



d) Error Contours

Figure 11-16. Example of Oscillations Induced by the Multiquadrics Method (Test 1p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2 (X-axis and Y-axis have been expanded for visibility)

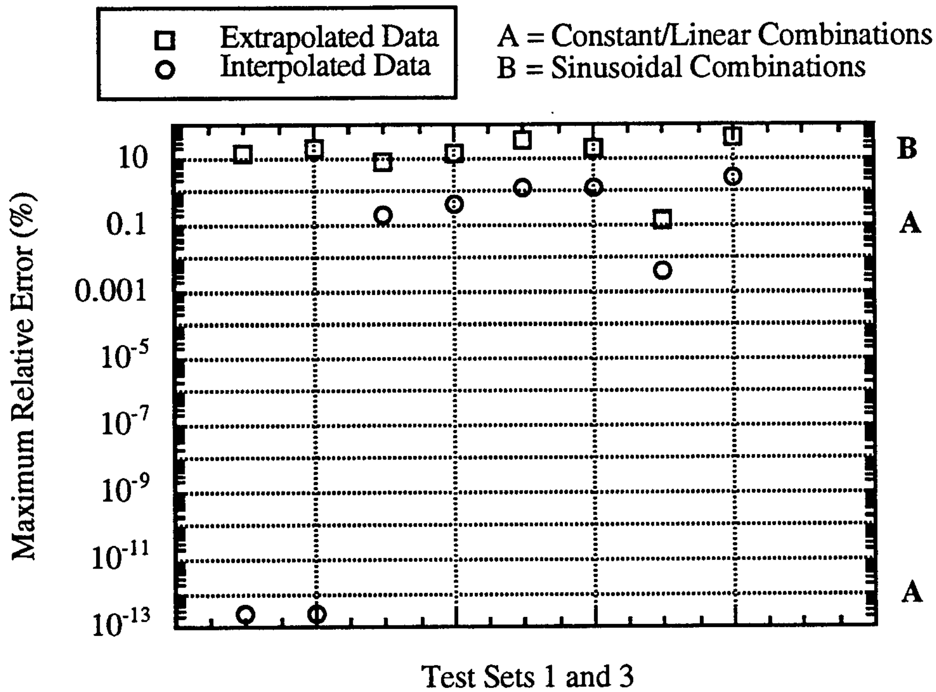


Figure 11-17. Variation of Error for Interpolation (Test Set 1) and Extrapolation (Test Set 3) Using Multiquadrics

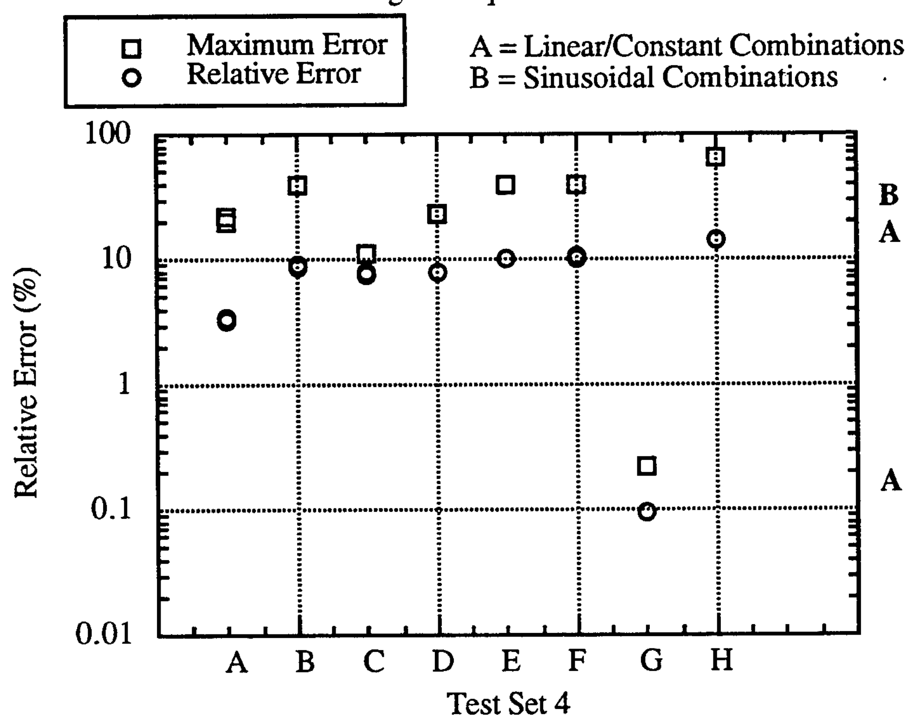


Figure 11-18. Variation of Error for Increasing Grid Fineness (Test Set 4) Using Multiquadrics

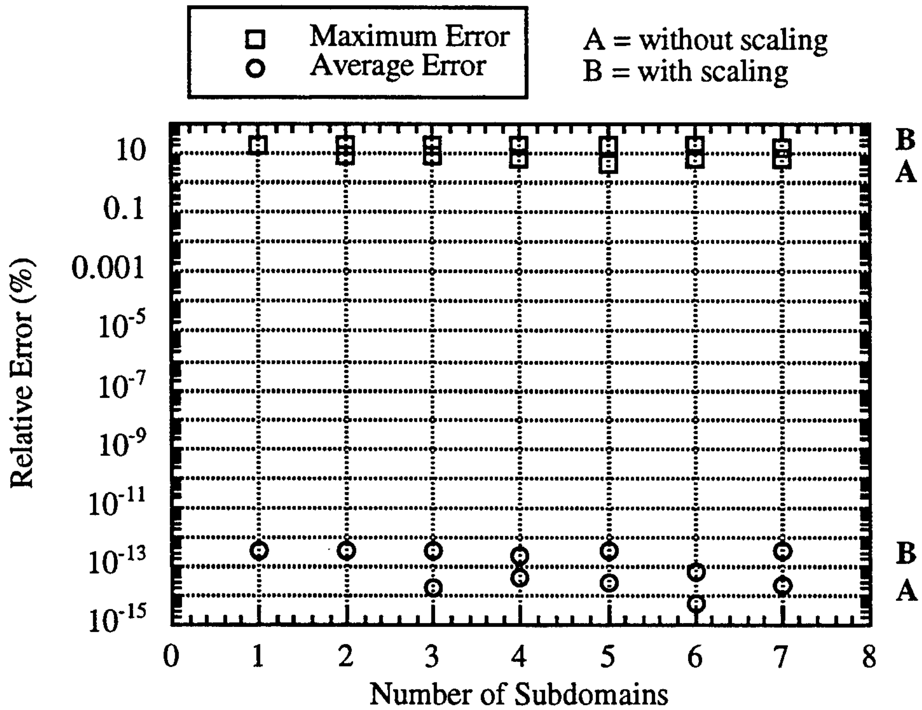


Figure 11-19. Variation of the Maximum Error as Functions of the Number of Subdomains and Scaling for the Multiquadrics Method (Test 1p)

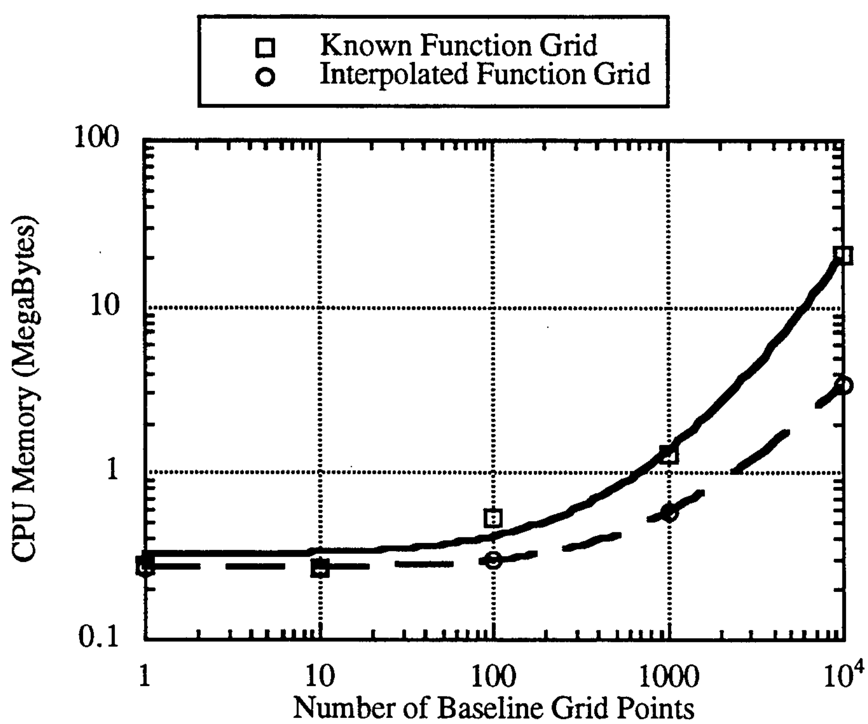


Figure 11-20. CPU Memory Requirements for the Multiquadrics Method as Implemented with 10% Overlapping and No More Than 11-20 Points Along Each Direction of the Subdomain

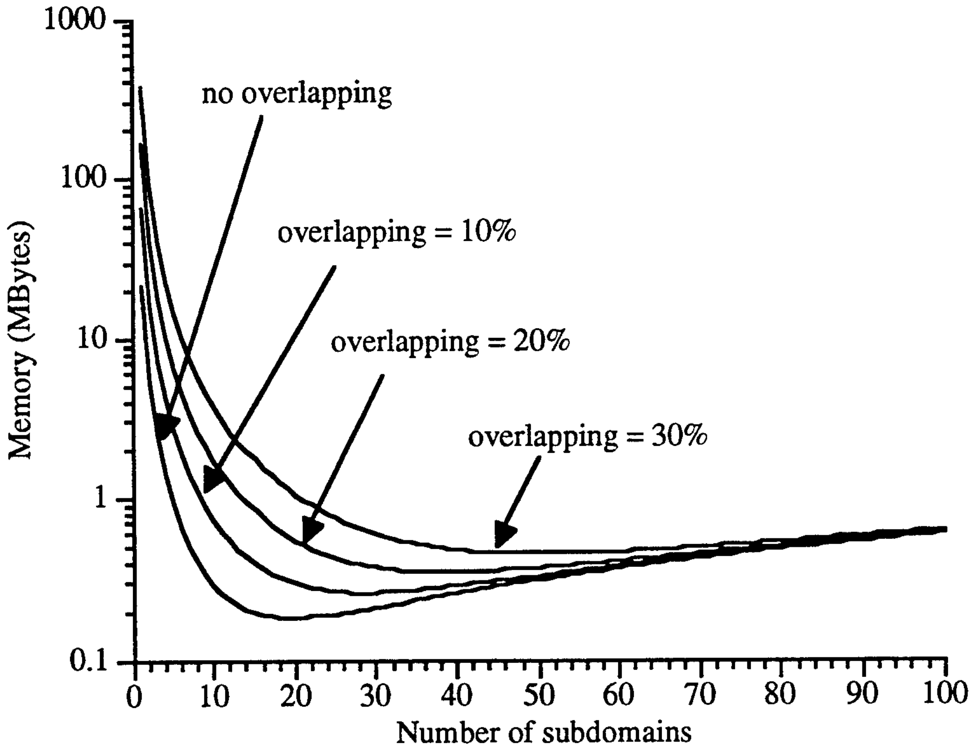


Figure 11-21. Influence of the Size of the Overlapping Region and the Subdomaining on the Memory Requirement for a Hypothetical Test Case (input grid = 150 x 150; output grid = 200 x 200) for the Multiquadrics Method

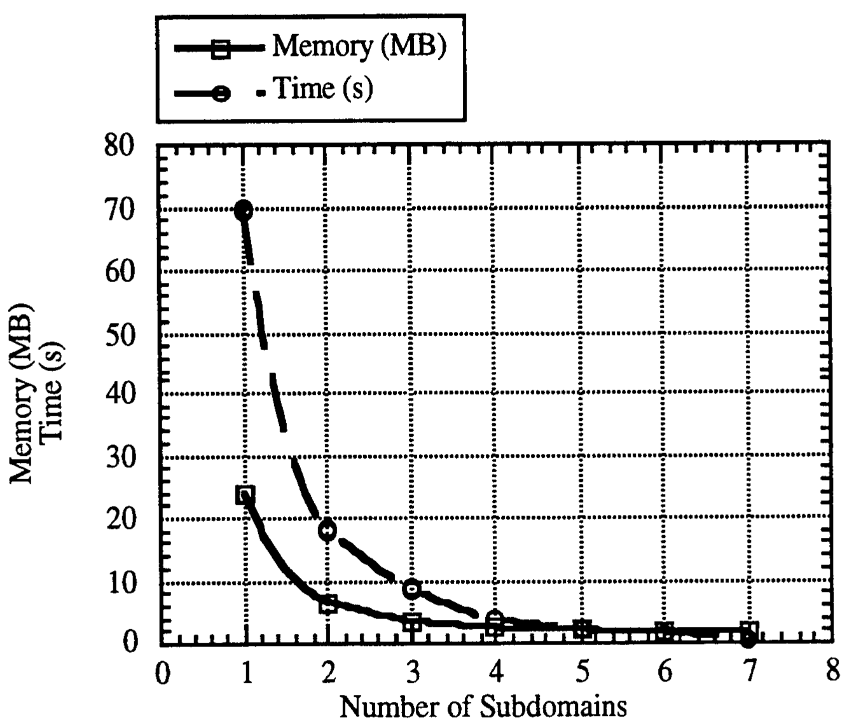


Figure 11-22. Influence of Subdomaining on the Memory Requirement and CPU Time for Multiquadrics Method (Test 1p)

Table 11.6 Statistical Summary of Test Set 1 using Multiquadrics

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	5.33E-15	1.07E-13	4	1	1.63E-15	3.25E-14	1.66E-15	1.165
test1b	8.88E-15	1.78E-13	4	1	1.64E-15	3.29E-14	2.06E-15	1.455
test1c	8.66E-15	4.40E-13	9	1	1.68E-15	8.53E-14	2.26E-15	1.54125
test1d	5.33E-15	1.07E-13	40	1	9.88E-16	1.98E-14	1.63E-15	0.3375
test1e	2.15E-02	0.4301	24	12	5.06E-03	0.1012	1.07E-02	0.69688
test1f	8.03E-02	4.024	47	1	2.55E-02	1.28	3.03E-02	0.87313
test1g	4.44E-15	8.88E-14	11	1	2.74E-15	5.47E-14	9.60E-04	1.02094
test1h	2.18E-02	0.4351	27	9	5.24E-03	0.1047	6.21E-03	1.06234
test1i	8.03E-02	4.024	4	1	2.53E-02	1.27	4.71E-02	1.13125
test1j	4.44E-15	8.88E-14	22	1	1.93E-15	3.85E-14	1.49E-03	1.15922
test1k	2.11E-02	0.4219	38	15	5.09E-03	0.1017	8.76E-03	1.22734
test1l	7.88E-02	3.966	21	1	2.59E-02	1.302	4.01E-02	1.29297
test1m	1.07E-13	2.13E-13	7	1	2.20E-14	4.41E-14	4.03E-03	2.64985
test1n	2.00E-15	1.96E-13	4	1	4.57E-16	4.48E-14	4.05E-04	2.66508
test1o	2.43E-17	1.21E-13	10	2	7.75E-18	3.88E-14	4.07E-05	2.65031
test1p	0.3551	17.79	1	2	0.1056	5.293	0.1694	8.4382
test1q	0.7957	39.79	85	2	0.2735	13.68	0.4424	9.56359
test1s	3.55E-03	17.79	1	2	1.06E-03	5.293	1.20E-02	8.0166
test1t	7.96E-03	39.79	85	2	2.74E-03	13.68	4.43E-03	9.44926
test1a1	7.63E-17	1.53E-13	10	3	1.80E-17	3.59E-14	4.45E-04	3.02394
test1b1	5.55E-17	1.11E-13	4	1	1.24E-17	2.48E-14	4.48E-05	3.00656
test1c1	1.87E-16	3.81E-13	4	2	4.91E-17	9.97E-14	4.50E-06	3.03925
test1d1	6.94E-17	1.39E-13	40	1	1.88E-17	3.77E-14	1.42E-07	1.703
test1e1	2.15E-04	0.4301	24	12	5.06E-05	0.1012	1.07E-04	1.75781
test1f1	7.88E-04	3.943	15	1	2.21E-04	1.103	3.45E-04	1.71254
test1g1	6.94E-17	1.39E-13	49	2	2.09E-17	4.19E-14	1.09E-05	1.75742
test1h1	2.18E-04	0.4351	27	9	5.24E-05	0.1047	6.21E-05	1.75214
test1i1	8.03E-04	4.024	4	1	2.53E-04	1.27	4.71E-04	1.74699
test1j1	6.94E-17	1.39E-13	48	8	1.75E-17	3.49E-14	1.49E-05	1.77191
test1k1	2.11E-04	0.4219	38	15	5.09E-05	0.1017	8.76E-05	1.80336
test1l1	7.88E-04	3.966	21	1	2.59E-04	1.302	4.01E-04	1.8208
test1a2	5.33E-15	1.07E-13	4	1	1.63E-15	3.25E-14	1.66E-15	1.145
test1b2	1.78E-14	3.55E-13	3	1	2.92E-15	5.84E-14	2.88E-15	1.435
test1c2	2.84E-14	1.44E-12	9	4	4.17E-15	2.12E-13	6.21E-15	1.5525
test1d2	5.33E-15	1.07E-13	40	1	9.88E-16	1.98E-14	1.64E-15	0.3325
test1e2	2.13E-02	0.4256	24	12	5.29E-03	0.1059	1.09E-02	0.6875
test1f2	7.53E-02	3.765	1	19	2.53E-02	1.263	3.47E-02	0.86438
test1g2	4.44E-15	8.88E-14	11	1	2.74E-15	5.47E-14	1.10E-03	1.00156
test1h2	2.15E-02	0.4306	27	9	5.45E-03	0.1089	6.58E-03	1.09266
test1i2	7.57E-02	3.785	1	2	2.51E-02	1.257	4.38E-02	1.13953
test1j2	4.44E-15	8.88E-14	22	1	1.93E-15	3.85E-14	1.39E-03	1.17813
test1k2	2.07E-02	0.4149	38	15	5.35E-03	0.1069	8.98E-03	1.25656
test1l2	7.46E-02	3.739	1	9	2.75E-02	1.377	4.23E-02	1.31234
test1m2	1.07E-13	2.13E-13	7	1	2.20E-14	4.41E-14	4.25E-03	2.64914
test1n2	2.44E-15	2.40E-13	7	1	6.46E-16	6.34E-14	4.27E-04	2.64453
test1o2	2.43E-17	1.21E-13	4	1	3.59E-18	1.80E-14	4.29E-05	2.64977
test1p2	0.714	35.82	1	2	0.207	10.39	0.3566	8.29977
test1a12	7.63E-17	1.53E-13	10	3	1.80E-17	3.59E-14	2.18E-17	1.135
test1b12	6.94E-17	1.39E-13	3	1	2.10E-17	4.19E-14	2.24E-17	1.44
test1c12	3.89E-16	7.90E-13	2	1	9.79E-17	1.99E-13	1.53E-16	1.5475
test1d12	6.94E-17	1.39E-13	40	1	1.88E-17	3.77E-14	2.63E-17	0.3625
test1e12	2.13E-04	0.4256	24	12	5.29E-05	0.1059	1.09E-04	0.6925
test1f12	7.54E-04	3.783	1	6	2.26E-04	1.132	3.51E-04	0.88313
test1g12	6.94E-17	1.39E-13	49	2	2.09E-17	4.19E-14	1.11E-05	0.96937
test1h12	2.15E-04	0.4306	27	9	5.45E-05	0.1089	6.58E-05	1.08438
test1i12	7.57E-04	3.785	1	2	2.51E-04	1.257	4.38E-04	1.12203
test1j12	6.94E-17	1.39E-13	48	8	1.75E-17	3.49E-14	1.39E-05	1.1214
test1k12	2.07E-04	0.4149	38	15	5.35E-05	0.1069	8.98E-05	1.26094
test1l12	7.46E-04	3.739	1	9	2.75E-04	1.377	4.23E-04	1.28453

Table 11.6 Statistical Summary of Test Set 1 using Multiquadrics (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1aa	5.00E-07	9.69E-06	7	5	1.80E-07	3.49E-06	1.81E-07	1.155
test1bb	8.84E-07	1.77E-05	7	4	2.03E-07	4.07E-06	2.05E-07	1.4225
test1cc	6.31E-07	3.08E-05	4	2	1.35E-07	6.60E-06	1.38E-07	1.55125
test1dd	7.21E-07	1.39E-05	19	9	2.24E-07	4.31E-06	2.24E-07	0.3425
test1ee	2.17E-02	0.4334	24	12	5.10E-03	0.1021	1.03E-02	0.69625
test1ff	8.10E-02	3.864	47	1	2.57E-02	1.224	6.05E-02	0.8675
test1gg	7.21E-07	1.39E-05	32	12	2.24E-07	4.31E-06	1.91E-03	0.99125
test1hh	2.19E-02	0.4385	27	9	5.28E-03	0.1056	1.00E-02	1.07609
test1ii	8.10E-02	3.904	4	1	2.55E-02	1.228	5.93E-02	1.07297
test1jj	6.52E-07	1.26E-05	17	11	2.34E-07	4.54E-06	1.88E-03	1.18
test1kk	2.12E-02	0.4248	38	15	5.13E-03	0.1026	1.05E-02	1.24625
test1ll	7.94E-02	3.827	21	1	2.60E-02	1.254	6.21E-02	1.31328
test1mm	4.70E-06	9.37E-06	5	6	1.47E-06	2.94E-06	6.24E-03	2.64227
test1nn	8.29E-07	7.06E-05	5	9	2.35E-07	2.00E-05	6.28E-04	2.64734
test1oo	6.60E-08	3.80E-05	4	5	8.05E-09	4.64E-06	6.31E-05	2.65273
test1pp	0.3551	16.48	1	2	0.1065	4.945	0.324	8.30594
test1qq	0.8488	39.45	46	2	0.2745	12.76	0.6422	9.53258
test1ss	3.55E-03	1.875	1	2	1.07E-03	0.5624	1.74E-02	8.12167
test1tt	8.49E-03	4.372	46	2	2.75E-03	1.414	6.43E-03	9.36453
test1bb2	8.88E-15	1.78E-13	4	1	1.64E-15	3.29E-14	2.05E-15	1.145
test1cc2	8.66E-15	4.40E-13	9	1	1.68E-15	8.53E-14	2.26E-15	1.46
test1dd2	5.33E-15	1.07E-13	40	1	9.88E-16	1.98E-14	1.63E-15	0.2775
test1ee2	2.15E-02	0.4301	24	12	5.06E-03	0.1012	1.07E-02	0.65188
test1ff2	8.03E-02	4.024	47	1	2.55E-02	1.28	3.03E-02	0.85375
test1gg2	4.44E-15	8.88E-14	11	1	2.74E-15	5.47E-14	9.60E-04	0.96281
test1hh2	2.18E-02	0.4351	27	9	5.24E-03	0.1047	6.21E-03	1.08453
test1ii2	8.03E-02	4.024	4	1	2.53E-02	1.27	4.71E-02	1.11281
test1jj2	4.44E-15	8.88E-14	22	1	1.93E-15	3.85E-14	1.49E-03	1.19141
test1kk2	2.11E-02	0.4219	38	15	5.09E-03	0.1017	8.76E-03	1.26937
test1ll2	7.88E-02	3.966	21	1	2.59E-02	1.302	4.01E-02	1.2975
test1mm2	1.07E-13	2.13E-13	7	1	2.20E-14	4.41E-14	4.03E-03	2.6457
test1nn2	2.00E-15	1.96E-13	4	1	4.57E-16	4.48E-14	4.05E-04	2.65055
test1oo2	2.43E-17	1.21E-13	10	2	7.75E-18	3.88E-14	4.07E-05	2.64578
test1pp2	0.3551	17.79	1	2	0.1056	5.293	0.1694	8.3232
test1qq2	0.7957	39.79	85	2	0.2735	13.68	0.4424	9.58891
test1ss2	3.55E-03	17.79	1	2	1.06E-03	5.293	1.20E-02	8.1089
test1tt2	7.96E-03	39.79	85	2	2.74E-03	13.68	4.43E-03	9.40129
test1A	2.31E-14	2.31E-13	2	1	5.03E-15	5.03E-14	9.43E-15	1.155
test1B	1.91E-14	2.74E-13	10	3	4.76E-15	6.83E-14	5.85E-15	1.435
test1C	2.11E-02	0.211	38	15	5.09E-03	5.09E-02	8.76E-03	0.2325
test1D	4.18E-02	0.4184	38	15	6.98E-03	6.98E-02	1.23E-02	0.58437
test1E	8.89E-02	1.272	21	1	2.64E-02	0.3779	4.10E-02	0.80844
test1F	8.41E-02	1.203	1	9	2.78E-02	0.397	4.30E-02	0.98875
test1G	8.34E-05	4.09E-03	38	15	1.40E-05	6.84E-04	1.36E-03	0.97766
test1H	7.46E-02	2.473	1	9	2.75E-02	0.9112	4.23E-02	0.95047
test1I	0.4078	10.21	66	1	0.1127	2.822	0.1808	8.30125
test1J	0.2702	13.38	53	64	2.39E-02	1.184	5.23E-02	227.60141
test1A1	8.33E-07	8.33E-06	6	9	2.37E-07	2.37E-06	2.38E-07	1.175
test1B1	8.29E-07	1.19E-05	3	8	1.75E-07	2.51E-06	1.77E-07	1.4725
test1C1	2.12E-02	0.2124	38	15	5.13E-03	5.13E-02	5.14E-03	0.26625
test1D1	4.21E-02	0.4212	38	15	7.04E-03	7.04E-02	7.04E-03	0.6275
test1E1	8.93E-02	1.278	21	1	2.66E-02	0.3802	2.66E-02	0.79813
test1F1	8.45E-02	1.209	1	9	2.79E-02	0.3986	2.79E-02	0.83406
test1G1	8.44E-05	3.86E-03	38	15	1.41E-05	6.44E-04	8.83E-04	1.02125
test1H1	7.52E-02	2.418	1	9	2.76E-02	0.8869	2.76E-02	1.05828
test1I1	0.4079	10.01	66	1	0.1137	2.788	0.1137	8.49703
test1J1	0.2612	11.98	53	64	2.40E-02	1.1	2.40E-02	228.25999

Table 11.6 Statistical Summary of Test Set 1 using Multiquadrics (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1A2	0.3062	3.062	3	3	0.1719	1.719	0.1728	3.3772
test1B2	1.611	22.49	7	8	0.7693	10.74	0.7733	3.37787
test1C2	0.1528	1.499	1	10	3.62E-02	0.3552	4.37E-02	2.03885
test1D2	0.1702	1.702	1	10	5.99E-02	0.5987	5.99E-02	2.03247
test1E2	1.266	17.75	21	10	0.2524	3.537	0.2525	2.04605
test1F2	1.262	17.67	27	12	0.2401	3.363	0.2404	2.02963
test1G2	3.41E-04	1.41E-02	1	10	1.20E-04	4.94E-03	7.61E-03	2.03336
test1H2	1.251	37.28	25	12	0.2394	7.136	0.2396	2.077
test1I2	3.584	82.8	50	12	1.306	30.17	1.306	7.58902
test1J2	3.141	130.3	53	56	1.289	53.46	1.054	226.86172

Table 11.7 Statistical Summary of Test Set 2 using Multiquadrics

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test2A	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	1.42
test2B	2.27E-02	1.134	5	1	4.07E-03	0.2036	7.02E-03	1.65
test2C	1.68E-02	0.8401	1	4	4.89E-03	0.2444	7.24E-03	1.7375
test2D	0.1125	1.875	5	1	2.15E-02	0.3588	3.55E-02	1.7975
test2E	0.8923	45.3	8	7	0.2421	12.29	0.4196	1.84
test2F	0.00E+00	0.00E+00	8	7	0.00E+00	0.00E+00	8.39E-03	1.02125
test2G	2.26E-02	1.128	25	1	3.33E-03	0.1663	5.57E-03	1.48687
test2H	1.72E-02	0.8619	1	22	3.94E-03	0.1969	5.80E-03	1.66281
test2I	0.1117	1.862	25	1	1.71E-02	0.2842	2.72E-02	1.84687
test2L	1.31E-04	1.308	33	32	2.45E-05	0.245	5.45E-04	1.92719
test2M	1.18E-04	1.181	25	1	2.70E-05	0.27	4.49E-05	1.98672
test2N	9.05E-04	45.56	32	30	1.66E-04	8.336	3.14E-04	2.03727
test2A1	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	1.39
test2B1	2.27E-02	1.134	5	1	3.79E-03	0.1896	3.81E-03	1.595
test2C1	1.68E-02	0.8401	1	4	4.65E-03	0.2325	4.69E-03	1.7175
test2D1	0.1125	1.875	5	1	1.97E-02	0.3277	1.98E-02	1.77
test2E1	0.9201	45.8	8	7	0.2622	13.05	0.2636	1.83625
test2F1	6.06E-07	2.78E-05	10	7	2.36E-07	1.09E-05	5.27E-03	1.00625
test2G1	2.26E-02	1.128	25	1	3.25E-03	0.1626	3.25E-03	1.48938
test2H1	1.72E-02	0.8619	1	22	3.86E-03	0.1932	3.87E-03	1.63406
test2I1	0.1117	1.862	25	1	1.67E-02	0.278	1.67E-02	1.79516
test2L1	1.28E-04	6.95E-02	33	32	2.39E-05	1.30E-02	3.35E-04	1.90141
test2M1	1.18E-04	6.43E-02	25	1	2.65E-05	1.44E-02	2.73E-05	1.95922
test2N1	9.30E-04	0.5203	31	29	1.71E-04	9.54E-02	1.70E-04	2.03867

Table 11.8 Statistical Summary of Test Set 3 using Multiquadrics

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A	0.7014	14.03	1	1	5.43E-02	1.085	0.1186	1.135
test3B	0.6772	19.43	1	3	9.34E-02	2.68	0.22	1.4625
test3C	0.3856	7.712	1	1	9.94E-02	1.988	0.1364	0.285
test3D	0.7014	14.03	1	1	0.1205	2.409	0.1855	0.64062
test3E	1.208	34.57	1	19	0.2851	8.163	0.6586	0.83563
test3F	0.7032	20.11	1	9	0.1757	5.024	0.3384	0.99094
test3G	1.40E-03	0.1375	1	1	2.41E-04	2.36E-02	1.07E-02	1.08891
test3H	0.5548	36.81	1	12	0.1762	11.69	0.3783	1.06656
test3I	2.296	115	1	6	0.2274	11.39	0.6219	9.41875
test3J	1.807	179	4	48	0.2056	20.36	0.464	355.88361
test3A3	0.00E+00	0.00E+00	4	48	0.00E+00	0.00E+00	4.66E-02	3.85333
test3B3	9.39E-02	4.694	10	1	1.56E-02	0.7791	3.13E-02	3.85358
test3C3	9.49E-02	4.746	10	10	1.41E-02	0.7054	3.06E-02	3.85382
test3D3	0.4481	7.468	1	1	7.72E-02	1.286	0.1508	3.8541
test3E3	1.1	55.83	10	10	0.2711	13.76	0.4486	3.85437
test3F3	0.00E+00	0.00E+00	10	10	0.00E+00	0.00E+00	8.97E-03	2.91547
test3G3	9.39E-02	4.694	50	1	3.31E-02	1.654	5.31E-02	2.91089
test3H3	9.49E-02	4.746	50	50	3.02E-02	1.508	5.52E-02	2.9166
test3I3	0.5495	9.158	49	50	0.167	2.784	0.2744	2.93231
test3J3	1.1	55.36	50	50	0.3316	16.69	0.5076	2.91797
test3K3	5.20E-18	5.20E-14	19	19	8.50E-19	8.50E-15	1.02E-02	2.9136
test3L3	9.26E-04	9.264	50	49	1.94E-04	1.937	4.22E-04	2.94916
test3M3	9.38E-04	9.376	49	1	1.95E-04	1.954	3.83E-04	2.94473
test3N3	1.10E-03	55.36	50	50	3.31E-04	16.68	5.08E-04	2.97037
test3A1	1.00E-06	2.01E-05	4	3	2.96E-07	5.91E-06	2.97E-07	1.015
test3B1	0.4064	11.66	1	8	5.64E-02	1.619	0.302	1.3375
test3C1	1.52E-06	3.04E-05	9	13	4.13E-07	8.26E-06	9.56E-03	0.4075
test3D1	1.14E-06	2.28E-05	7	2	2.77E-07	5.54E-06	3.02E-04	0.01563
test3E1	0.8592	24.6	1	20	0.2373	6.794	0.3373	0.16875
test3F1	0.4098	11.72	1	12	9.59E-02	2.742	0.3462	0.30406
test3G1	1.24E-06	1.06E-04	4	16	3.62E-07	3.09E-05	1.10E-02	0.34516
test3H1	0.4098	25.55	1	12	9.59E-02	5.979	9.59E-02	0.44328
test3I1	2.34	112.4	1	1	0.2773	13.32	2.618	12.32219
test3J1	4.417	374.8	3	84	0.5241	44.47	2.735	409.80814
test3A2	1.07E-06	2.13E-05	4	6	3.13E-07	6.25E-06	0.2749	3.47241
test3B2	0.4117	11.81	1	8	5.44E-02	1.562	0.2931	3.48315
test3C2	1.56E-06	3.11E-05	9	13	4.24E-07	8.48E-06	9.27E-03	1.72421
test3D2	1.18E-06	2.37E-05	7	2	2.86E-07	5.72E-06	2.93E-04	1.74835
test3E2	0.8659	24.79	1	7	0.2363	6.764	0.3374	1.73242
test3F2	0.4146	11.85	1	12	9.44E-02	2.7	0.3363	1.7265
test3G2	1.31E-06	1.12E-04	8	10	3.74E-07	3.19E-05	1.06E-02	1.74057
test3H2	0.4146	25.85	1	12	9.44E-02	5.887	9.45E-02	1.75467
test3I2	2.364	113.5	1	1	0.2334	11.21	2.598	10.10089
test3J2	4.631	392.9	3	49	0.4952	42.02	2.859	396.13104

Table 11.8 Statistical Summary of Test Set 3 using Multiquadrics (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A4	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	0.2874	4.3399
test3B4	5.49E-07	2.75E-05	7	8	1.93E-07	9.64E-06	2.89E-02	4.34027
test3C4	5.49E-07	2.75E-05	8	7	1.93E-07	9.65E-06	2.90E-03	4.34058
test3D4	9.75E-07	1.63E-05	3	3	2.35E-07	3.92E-06	2.92E-04	4.35083
test3E4	0.7094	35.31	10	10	0.1458	7.259	0.5398	4.34113
test3F4	6.06E-07	2.78E-05	10	7	2.36E-07	1.09E-05	1.08E-02	3.16223
test3G4	6.06E-07	3.03E-05	23	16	2.39E-07	1.20E-05	2.16E-04	3.15802
test3H4	6.04E-07	3.02E-05	16	23	2.39E-07	1.20E-05	4.33E-06	3.20374
test3I4	1.02E-06	1.70E-05	48	18	2.50E-07	4.17E-06	2.60E-07	3.14948
test3J4	0.7094	34.71	50	50	0.185	9.049	0.3957	3.18536
test3K4	4.78E-08	2.54E-05	28	29	6.08E-10	3.23E-07	7.92E-03	3.18115
test3L4	9.74E-08	5.30E-05	25	28	1.21E-09	6.60E-07	1.58E-04	3.21716
test3M4	9.73E-08	5.30E-05	28	26	1.22E-09	6.64E-07	3.17E-06	3.18298
test3N4	7.09E-04	0.3968	50	50	1.85E-04	0.1035	3.57E-04	3.21881
test3A1	0.7055	14.11	1	1	5.68E-02	1.137	5.71E-02	1.1
test3B1	0.6738	19.33	1	3	9.85E-02	2.826	0.6708	1.4475
test3C1	0.3881	7.763	1	1	0.1007	2.013	0.1029	0.285
test3D1	0.7055	14.11	1	1	0.1221	2.441	0.1222	0.65125
test3E1	1.205	34.49	1	20	0.2887	8.265	0.4369	0.82375
test3F1	0.7005	20.03	1	9	0.1782	5.097	0.7609	0.94656
test3G1	1.41E-03	0.1204	1	1	2.44E-04	2.08E-02	2.41E-02	1.05469
test3H1	0.5507	34.34	1	12	0.1787	11.14	0.1788	1.11219
test3I1	2.306	110.8	1	6	0.2636	12.66	2.254	9.35563
test3J1	1.745	148	4	83	0.2934	24.9	0.2211	358.60812
test3A2	0.7014	14.03	1	1	5.83E-02	1.167	6.27E-02	3.51489
test3B2	0.6772	19.43	1	3	0.1442	4.137	0.7087	3.52554
test3C2	0.3856	7.712	1	1	9.96E-02	1.991	0.1021	2.19653
test3D2	0.7014	14.03	1	1	0.1207	2.414	0.1208	2.19009
test3E2	1.209	34.6	1	20	0.2904	8.314	0.439	2.22369
test3F2	0.7032	20.11	1	9	0.1808	5.171	0.7568	2.22723
test3G2	1.40E-03	0.1197	1	1	2.42E-04	2.06E-02	2.40E-02	2.24075
test3H2	0.5548	34.6	1	12	0.1814	11.31	0.1815	2.22427
test3I2	2.296	110.3	1	6	0.3707	17.8	2.371	8.35175
test3J2	1.804	153	4	48	0.417	35.38	0.1062	357.65363
test3A4	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.07E-02	4.25964
test3B4	9.39E-02	4.694	10	1	1.53E-02	0.7631	7.66E-02	4.24994
test3C4	9.49E-02	4.746	10	10	1.37E-02	0.6854	8.20E-02	4.26019
test3D4	0.5521	9.201	10	10	7.77E-02	1.295	0.4787	4.25043
test3E4	1.1	54.73	10	10	0.2946	14.66	1.127	4.26074
test3F4	6.06E-07	2.78E-05	10	7	2.36E-07	1.09E-05	2.26E-02	3.33185
test3G4	9.39E-02	4.694	50	1	3.30E-02	1.652	5.84E-02	3.32721
test3H4	9.49E-02	4.746	50	50	3.01E-02	1.504	6.49E-02	3.3327
test3I4	0.5521	9.201	50	50	0.1668	2.78	0.3855	3.34814
test3J4	1.1	53.8	50	50	0.3363	16.46	0.7728	3.35364
test3K4	4.78E-08	2.54E-05	28	29	6.08E-10	3.23E-07	1.55E-02	3.34912
test3L4	9.32E-04	0.5072	50	50	1.93E-04	0.1052	8.01E-04	3.35455
test3M4	9.43E-04	0.5133	50	1	1.95E-04	0.1062	1.13E-03	3.33008
test3N4	1.10E-03	0.6152	50	50	3.36E-04	0.1881	7.00E-04	3.37567

Table 11.9 Statistical Summary of Test Set 4 using Multiquadrics

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test4A2	0.9922	19.84	2	3	0.1655	3.311	0.2407	3.86
test4A3	1.083	21.66	2	2	0.1732	3.464	0.251	21.8
test4B2	1.397	40.19	1	10	0.3051	8.778	0.4229	30.05
test4B3	1.4	40.25	1	19	0.3105	8.926	0.4235	48.23
test4C2	0.5637	11.27	3	1	0.393	7.859	0.2142	42.55
test4C3	0.5642	11.28	2	1	0.3813	7.626	0.2154	224.64
test4D2	1.128	22.56	3	1	0.3908	7.816	0.4326	42.5
test4D3	1.129	22.57	2	1	0.3894	7.787	0.4355	224.52
test4E2	1.402	40.29	145	1	0.3418	9.819	0.4181	42.36
test4E3	1.404	40.33	289	1	0.3423	9.828	0.4182	224.92999
test4F2	1.403	40.31	1	33	0.3562	10.23	0.4184	42.71
test4F3	1.398	40.22	1	65	0.3576	10.29	0.4195	225.09
test4G2	2.24E-03	0.2198	2	2	9.96E-04	9.77E-02	3.83E-03	313.10001
test4G3	2.25E-03	0.2204	400	147	9.92E-04	9.73E-02	1.97E-03	495.04999
test4H2	0.9608	64.41	200	33	0.2106	14.12	0.3292	42.78
test4H3	0.9551	64.25	400	65	0.2113	14.22	0.33	225.49001
test4A12	3.869	77.37	1	1	2.393	47.86	4.87	3.85
test4A13	3.869	77.37	1	1	2.395	47.91	4.869	22.5
test4B12	2.202	63.35	1	20	1.148	33.04	1.462	30.87
test4B13	2.202	63.3	1	40	1.146	32.95	1.196	49.62
test4C12	4.47	89.39	1	41	4.189	83.77	8.387	96.69
test4C13	4.47	89.39	1	81	4.19	83.8	8.389	279.01001
test4D12	3.869	77.37	1	1	2.923	58.45	5.906	367.07001
test4D13	3.869	77.37	1	1	2.925	58.49	5.908	548.58002
test4E12	2.202	63.27	174	1	1.281	36.8	2.629	636.47998
test4E13	2.202	63.23	348	1	1.282	36.8	2.619	226.82001
test4F12	2.201	63.2	1	40	1.785	51.27	1.854	42.63
test4F13	2.202	63.34	1	80	1.786	51.38	1.821	225.06
test4G12	1.018	84.47	200	80	1.014	84.15	2.028	313.07001
test4G13	1.018	84.15	400	160	1.014	83.82	2.028	494.70999
test4H12	0.775	49.1	176	22	0.4429	28.06	1.123	582.71002
test4H13	0.7753	49.29	352	42	0.4423	28.12	1.122	766.95001

Table 11.10 Statistical Summary of Test Set 5 using Multiquadrics with no Scaling

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test5a	1.94E-16	1.94E-14	4	1	0.00E+00	0.00E+00	2.13E-16	1.22
test5b	4.44E-16	4.44E-14	9	1	0.00E+00	0.00E+00	3.20E-16	1.42
test5c	6.66E-16	6.66E-14	4	1	0.00E+00	0.00E+00	2.61E-16	1.485
test5d	6.66E-16	6.66E-14	1	1	8.33E-17	8.33E-15	1.48E-16	1.31
test5e	1.31E-15	1.31E-13	11	1	0.00E+00	0.00E+00	5.96E-16	1.50875
test5f	2.61E-15	2.61E-13	82	1	0.00E+00	0.00E+00	8.67E-16	1.58
test5g	4.11E-04	4.11E-02	499	1	6.66E-16	6.66E-14	2.72E-05	1.01125
test5h	1.43E-03	0.1426	499	1	2.22E-15	2.22E-13	9.50E-05	1.26375
test5i	2.34E-03	0.2341	499	1	5.77E-15	5.77E-13	1.59E-04	1.425
test5j	1.04E-14	1.04E-11	1	1	0.00E+00	0.00E+00	5.30E-05	2.52125
test5k	3.47E-17	3.47E-14	9	1	0.00E+00	0.00E+00	1.77E-05	2.52375
test5l	6.94E-17	6.94E-14	9	1	0.00E+00	0.00E+00	5.89E-06	2.52594
test5m	5.55E-17	5.55E-14	1	1	0.00E+00	0.00E+00	5.92E-07	2.17312
test5n	1.68E-16	1.68E-13	82	1	2.93E-18	2.93E-15	5.95E-08	2.15594
test5o	3.47E-16	3.47E-13	74	1	0.00E+00	0.00E+00	5.98E-09	2.18062
test5p	4.11E-05	4.11E-02	499	1	0.00E+00	0.00E+00	2.72E-06	1.56188
test5k	1.53E-16	1.53E-13	1	1	0.00E+00	0.00E+00	9.07E-07	2.68062
test5r	2.34E-04	0.2341	499	1	1.39E-17	1.39E-14	1.59E-05	1.66469
test5a1	1.01E-15	3.38E-14	1	1	0.00E+00	0.00E+00	5.30E-06	2.72625
test5b1	1.33E-15	4.44E-14	3	1	0.00E+00	0.00E+00	1.77E-06	2.74797
test5c1	1.55E-15	5.18E-14	3	1	2.22E-16	7.40E-15	5.88E-07	2.74906
test5d1	3.55E-15	1.18E-13	33	1	0.00E+00	0.00E+00	5.91E-08	2.3625
test5e1	5.11E-15	1.70E-13	60	1	0.00E+00	0.00E+00	5.94E-09	2.38422
test5f1	5.33E-15	1.78E-13	48	1	0.00E+00	0.00E+00	5.97E-10	2.40578
test5g1	5.26E-04	1.75E-02	499	1	4.44E-15	1.48E-13	7.45E-05	1.75109
test5h1	9.68E-04	3.23E-02	499	1	1.73E-14	5.77E-13	1.39E-04	1.76719
test5i1	1.47E-03	4.89E-02	499	1	4.09E-14	1.36E-12	2.16E-04	1.83297
test5j1	5.77E-14	1.92E-11	1	1	0.00E+00	0.00E+00	7.19E-05	2.87328
test5k1	1.94E-16	6.48E-14	5	1	0.00E+00	0.00E+00	2.40E-05	2.89422
test5l1	4.16E-16	1.39E-13	3	1	0.00E+00	0.00E+00	7.99E-06	2.89484
test5m1	2.43E-16	8.10E-14	48	1	0.00E+00	0.00E+00	8.03E-07	2.51828
test5n1	1.11E-15	3.70E-13	38	1	6.07E-18	2.02E-15	8.07E-08	2.54016
test5o1	8.40E-16	2.80E-13	47	1	0.00E+00	0.00E+00	8.11E-09	2.54234
test5p1	1.90E-04	6.32E-02	499	1	1.11E-16	3.70E-14	1.25E-05	1.88828
test5k1	9.44E-16	3.15E-13	1	1	0.00E+00	0.00E+00	4.16E-06	2.96781
test5r1	5.52E-04	0.1839	499	1	0.00E+00	0.00E+00	3.71E-05	1.915

Table 11.11 Statistical Summary of Test Set 5 using Multiquadrics with Scaling

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test5a	1.94E-16	1.94E-14	4	1	0.00E+00	0.00E+00	2.13E-16	1.265
test5b	4.44E-16	4.44E-14	9	1	0.00E+00	0.00E+00	3.20E-16	1.44
test5c	6.66E-16	6.66E-14	4	1	0.00E+00	0.00E+00	2.61E-16	1.525
test5d	6.66E-16	6.66E-14	1	1	0.00E+00	0.00E+00	1.99E-16	1.33
test5e	2.00E-15	2.00E-13	62	1	0.00E+00	0.00E+00	1.13E-15	1.51
test5f	2.22E-15	2.22E-13	100	1	0.00E+00	0.00E+00	1.11E-15	1.58
test5g	4.11E-04	4.11E-02	499	1	1.33E-15	1.33E-13	2.72E-05	0.99375
test5h	1.43E-03	0.1426	499	1	1.33E-15	1.33E-13	9.50E-05	1.30187
test5i	2.34E-03	0.2341	499	1	6.22E-15	6.22E-13	1.59E-04	1.42437
test5j	1.02E-14	1.02E-11	1	1	0.00E+00	0.00E+00	5.30E-05	2.51125
test5k	3.47E-17	3.47E-14	9	1	0.00E+00	0.00E+00	1.77E-05	2.49406
test5l	6.94E-17	6.94E-14	9	1	0.00E+00	0.00E+00	5.89E-06	2.54719
test5m	5.55E-17	5.55E-14	1	1	0.00E+00	0.00E+00	5.92E-07	2.17375
test5n	1.42E-16	1.42E-13	12	1	0.00E+00	0.00E+00	5.95E-08	2.15656
test5o	3.40E-16	3.40E-13	64	1	0.00E+00	0.00E+00	5.98E-09	2.21
test5p	4.11E-05	4.11E-02	499	1	5.55E-17	5.55E-14	2.72E-06	1.54906
test5k	1.67E-16	1.67E-13	1	1	0.00E+00	0.00E+00	9.07E-07	2.66938
test5r	2.34E-04	0.2341	499	1	1.39E-17	1.39E-14	1.59E-05	1.61312
test5a1	7.63E-16	2.54E-14	1	1	0.00E+00	0.00E+00	5.30E-06	2.74641
test5b1	1.11E-15	3.70E-14	7	1	0.00E+00	0.00E+00	1.77E-06	2.72766
test5c1	1.89E-15	6.29E-14	6	1	0.00E+00	0.00E+00	5.88E-07	2.72859
test5d1	3.78E-15	1.26E-13	47	1	0.00E+00	0.00E+00	5.91E-08	2.35234
test5e1	7.33E-15	2.44E-13	47	1	0.00E+00	0.00E+00	5.94E-09	2.42406
test5f1	9.33E-15	3.11E-13	33	1	0.00E+00	0.00E+00	5.97E-10	2.38516
test5g1	5.52E-04	1.84E-02	499	1	0.00E+00	0.00E+00	7.40E-05	1.75031
test5h1	1.01E-03	3.38E-02	499	1	1.73E-14	5.77E-13	1.39E-04	1.77641
test5i1	1.54E-03	5.12E-02	499	1	3.42E-14	1.14E-12	2.13E-04	1.78312
test5j1	5.33E-14	1.78E-11	1	1	3.47E-18	1.16E-15	7.11E-05	2.87375
test5k1	1.11E-16	3.70E-14	6	1	1.39E-17	4.63E-15	2.37E-05	2.89516
test5l1	2.22E-16	7.40E-14	10	1	1.39E-17	4.63E-15	7.90E-06	2.87625
test5m1	4.23E-16	1.41E-13	43	1	1.39E-17	4.63E-15	7.94E-07	2.49984
test5n1	8.33E-16	2.78E-13	43	1	0.00E+00	0.00E+00	7.98E-08	2.51156
test5o1	1.12E-15	3.75E-13	39	1	1.39E-17	4.63E-15	8.02E-09	2.54375
test5p1	5.52E-05	1.84E-02	499	1	5.55E-17	1.85E-14	7.40E-06	1.87922
test5k1	1.72E-15	5.74E-13	1	1	1.39E-17	4.63E-15	2.47E-06	2.96969
test5r1	1.54E-04	5.12E-02	499	1	5.55E-17	1.85E-14	2.13E-05	1.93672

11.3 Non-Uniform B-Splines Method

The non-uniform B-splines method is the method which is based on an existing CAD package. This method is of particular interest since the CAD package (DT_NURBS) can conceivably be used for interfacing CAD geometries directly into any interface methodology. If the interface method and the CAD geometry packages can be combined as one CAD package, the memory savings and continuity across the code could be significant. The implementation of the non-uniform B-splines (NUBS) method is described in Chapter 6.

Overall Accuracy – Statistical summaries of these test cases are presented in Tables 11.12 to 11.15. The overall accuracy of the NUBS method is excellent, however the implementation of the method has some problems which need to be corrected in future applications. The primary shortcoming is that the correlation of the known and the unknown grids relies on a bilinear interpolation. This interpolation can introduce errors which are particularly felt for the double precision implementation. In addition, the search routine which locates the known structure panel (grid at which the function is known) in which the unknown grid point is located appears to have some regarding tolerances for points which are very close to panels, but because of the number of decimal points may not be "numerically" inside the panel.

For the cases which did not have any problems with the search routine, the overall accuracy of the NUBS method was excellent, as demonstrated by Figure 11-23. Figure 11-23 plots the error for each category of function test cases. It is easily observed from these figures that for all of the functions in test case 1, the method has excellent performance. For two-dimensional surfaces (plates), the maximum error which occurs is 5% with most errors less than 1%. For three-dimensional surfaces (shells), the maximum error increases to 25-30% for the very high frequency data. Again, most of the errors are less than 1% for the functions tested.

The very high accuracy of this algorithm is demonstrated by Figures 11-24 and 11-25 which are plots of one and three cycle sinusoidal functions, respectively. The errors are virtually zero, as seen in c) and d) for the two figures. In addition, the error remains consistent across the direction where the function remains constant. This is important since it demonstrates that the method is not adversely influenced by widely different functions in each direction. Functions of this type can be found in mode shapes and pressure coefficients.

Grid Spacing Sensitivity – Figure 11-23 indicates that the NUBS method is somewhat sensitive to grid spacing. In Figure 11-23, the data plotted on the first minor tick for each of the three function types is the regular grid, while the minor tics 2–4 in each function type correspond to the other clustered grids. For the constant functions, the interpolation was insensitive to the grid spacing to which the data was transferred. However, for the linear and sinusoidal functions, a dramatic difference was noted. The error increased between 8 to 12 orders of magnitude between these two types of grids. It should be noted that the errors are still below 10% of the overall magnitude or amplitude for all of the functions, and less than 1% of most of the functions.

Directional Bias – Figure 11-23 includes the streamwise (test1a – 11) and spanwise (test1a2 – 12) functions examined in test set 1. These data are virtually identical for both directions, indicating little or no sensitivity to direction of the function.

Magnitude/Amplitude Sensitivity – The NUBS method does not have any sensitivity to amplitude. In Figure 11-23, the error for magnitudes at 10^{-2} are two orders of magnitude less than functions whose magnitude is 10^{0}. This trend is consistent for function type, plates, shells, direction of the function and grid.

Sensitivity to Frequency (Higher Oscillations) – The NUBS method is somewhat sensitive to the frequency of the function. As shown in Figure 11-26, when the frequency of the sinusoidal function is increased to 3 and 5 cycles, the error of the NUBS method increases by two and one orders of magnitude, respectively. When the function is increased from 5 to 7 cycles, the error of the NUBS method decreases by over one order of magnitude. Overall, the error is still very small relative to the amplitude of the function (less than 0.1%).

Extrapolation – Test set 3 was designed to examine the algorithm's ability to extrapolate data. The maximum errors for test sets 1 and 3 are plotted in Figure 11-27 to illustrate the impact of extrapolation versus interpolation. It is apparent from this figure that the NUBS method is not as accurate for extrapolations since the maximum errors increase by several orders of magnitude which varies depending on the function type. It should be noted however, that the overall maximum errors remain less than 0.1% of the overall function amplitude or maximum value.

Diminishing Variation – Figure 11-28 shows that the error does not vary with increasing fineness, indicating that the interpolation scheme is consistent for each function.

Sensitivity to Three-Dimensional Surfaces (Plates Vs. Shells) – From Figure 11-23, it is apparent that the NUBS method does lose some accuracy when applied to the three-dimensional surfaces. The error increases by approximately 6 to 8 orders of magnitude. For the higher-frequency oscillatory functions, the error reaches almost 30% of the function amplitude. The remainder of the functions has errors of less than 5% (and in most instances is less than 1%).

Algorithm CPU Memory and Time Requirements – The average CPU time requirement varies between 1 to 10 seconds (refer to the last column of Tables 11.12 through 11.15), with the exception of the test cases which had very large errors, indicative of ill-conditioned matrices for several of the sinusoidal functions or problems with the search routine. These runs required on the average of 70 to 80 seconds.

The CPU memory requirements for NUBS are shown in Figure 11-29, computed using :

$$\begin{aligned} \text{CPU SIZE} = {} & 1.08\text{x}10^{6} + 190.79\text{KGS} - 0.00682\text{KGS}^{2} \\ & + 640.26\text{UKS} - 1.9605\text{x}10^{-5}\text{UKS}^{2} \end{aligned}$$

where KGS is the total number of grid points in the known function grid and UKS is the total number of grid points in the grid to be interpolated to. This function yields values within 2% of the actual CPU size during testing. The large starting memory size ($1.08\text{x}10^{6}$) is due to the requirement of a large 30,000 point work array. This memory requirement is expected to decrease with the implementation of dynamic memory routines in DT_NURBS 3.1.

Beam Element – The NUBS method requires that there be at least 4 points in each direction of the surface. Since the beam is modeled with one point, the NUBS methodology could not evaluate test set 5. It is possible to implement the DT_NURBS package to beam elements using different functions within the package.

Single Precision – The DT_NURBS package from which this method was implemented requires that the parameters be dimensioned as double precision. Without extensive modification to the entire CAD package – which was not a criteria of this research – evaluation of the method at single precision was not possible.

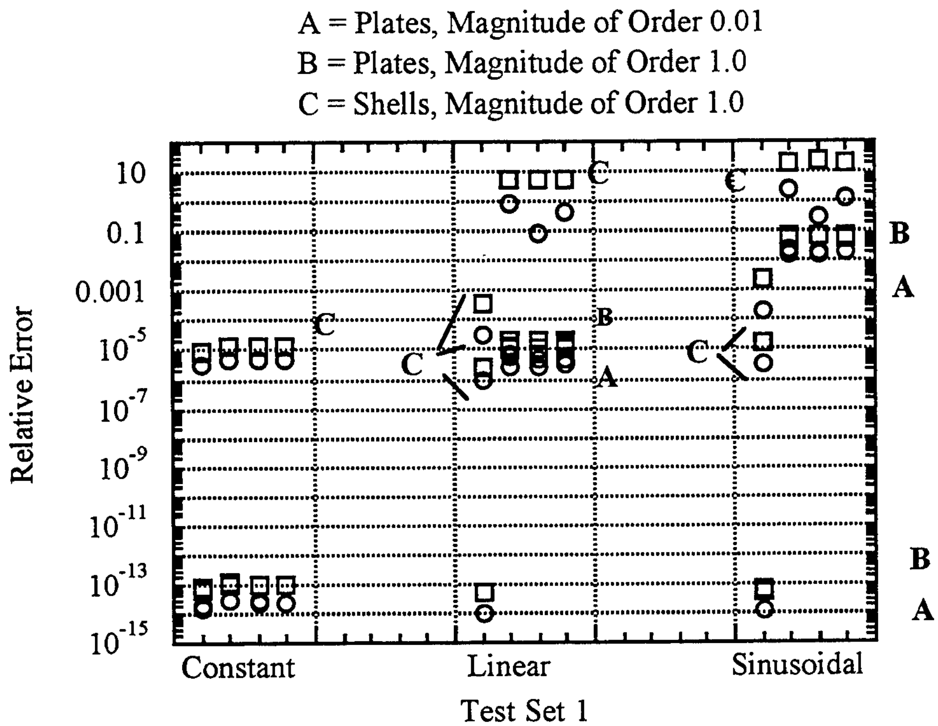
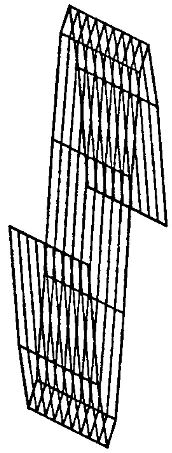
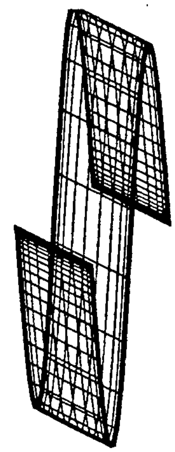


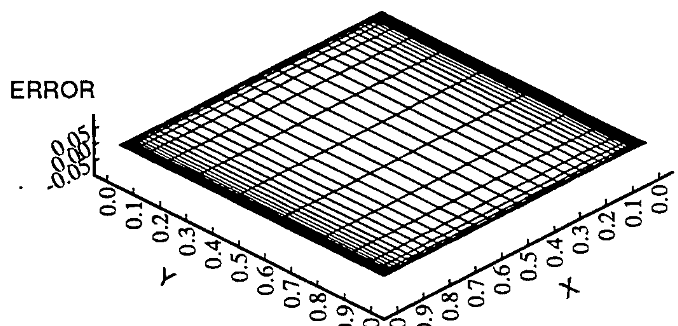
Figure 11-23. Variation of Error for Test Set One Based Upon Function Type for Plates Using the NUBS Method



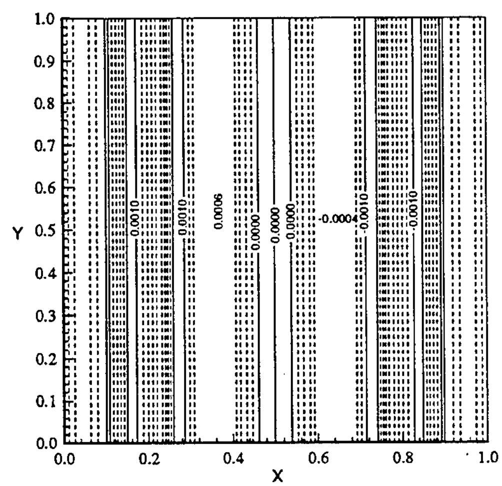
a) Original Function



b) Interpolated Function

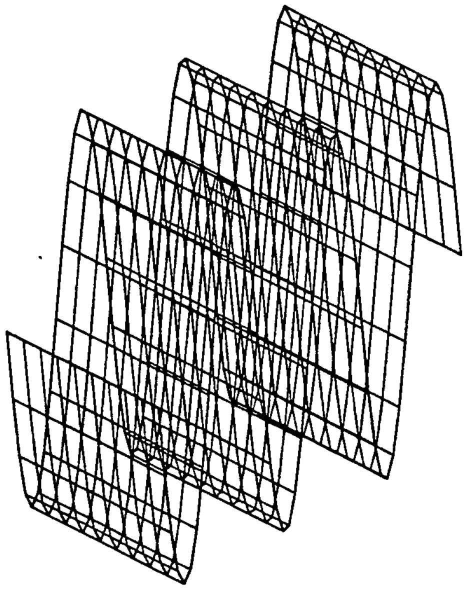


c) Orthogonal View of the Error

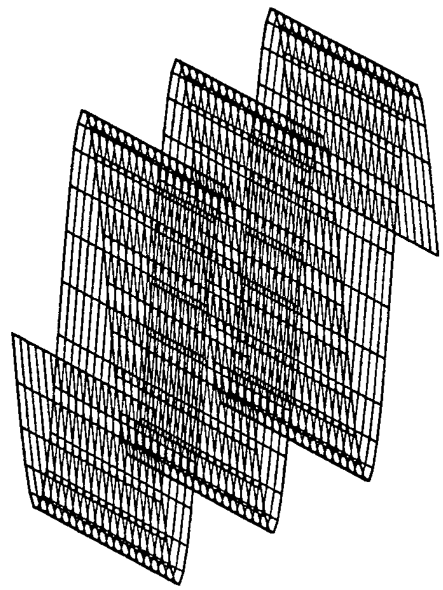


d) Error Contours

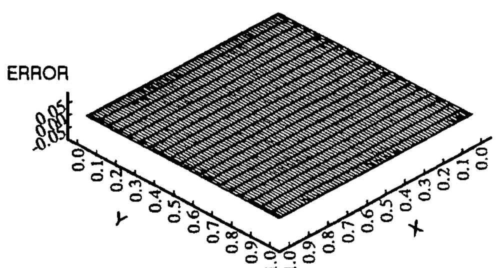
Figure 11-24. Example of Oscillations Induced by the NUBS Method (Test 1) for a One-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2



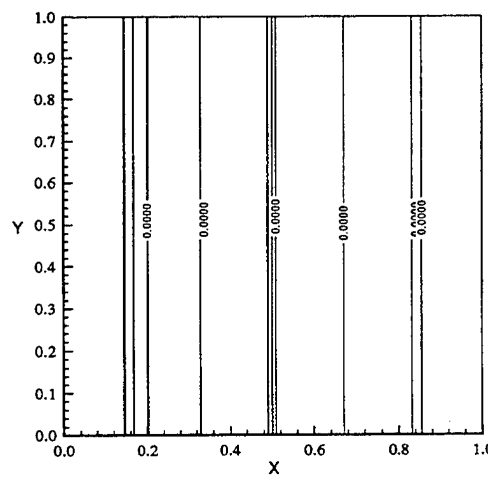
a) Original Function



b) Interpolated Function



c) Orthogonal View of Error



d) Error Contours

Figure 11-25. Example of Oscillations Induced by the NUBS Method (Test p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2 (X-axis and Y-axis have been expanded for visibility)

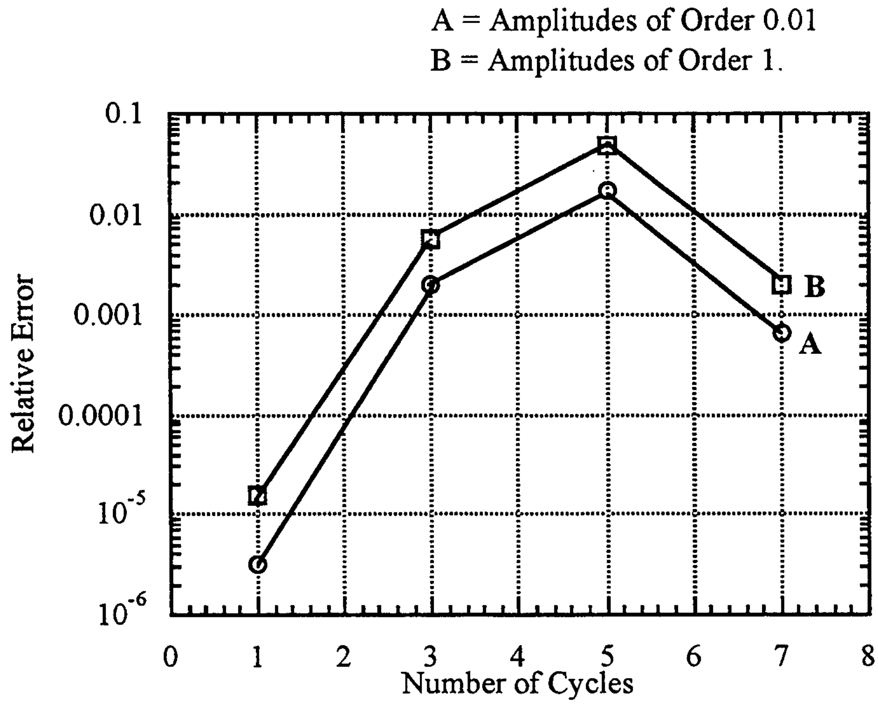


Figure 11-26. Variation of Error for Test Set One Based Upon Function Type for Plates Using the NUBS Method

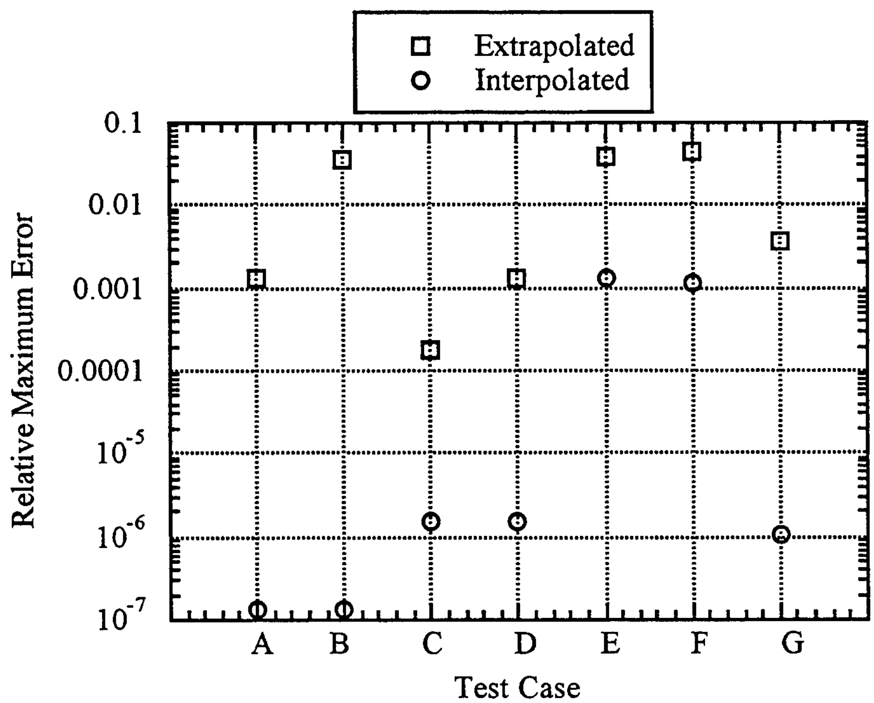


Figure 11-27. Variation of Error for Interpolation (Test Set 1) and Extrapolation (Test Set 3) For the NUBS Method

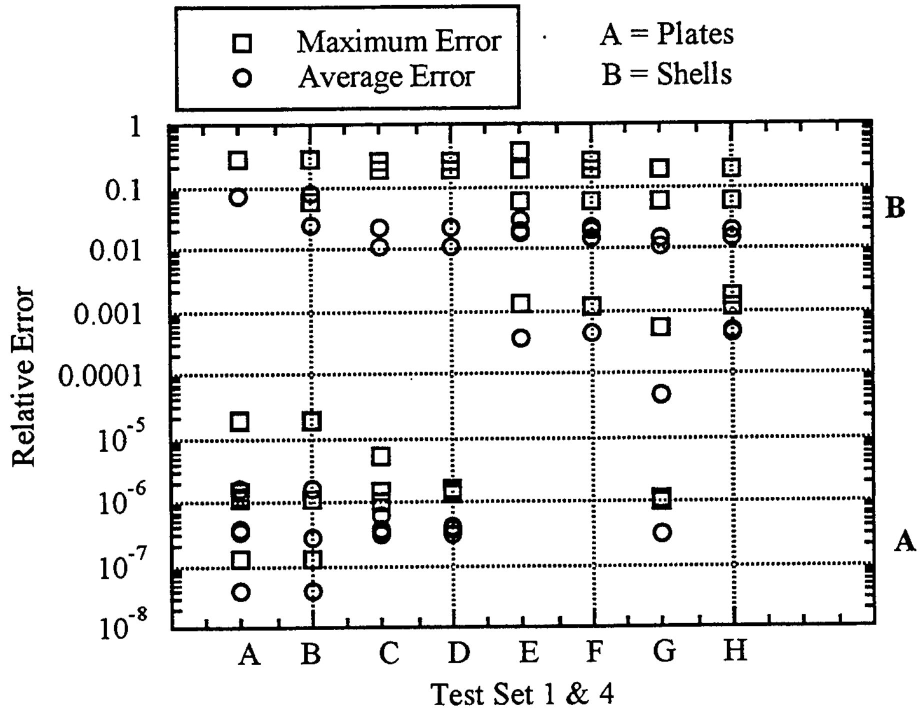


Figure 11-28. Variation of Error for Increasing Grid Fineness (Test Set 1 and 4) For the NUBS Method

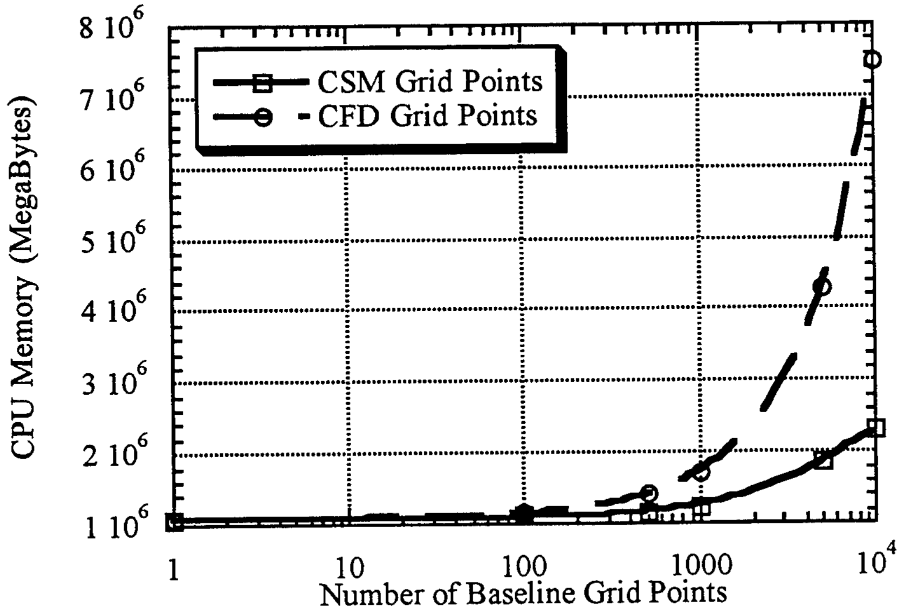


Figure 11-29. CPU Memory Requirements for the Non-Uniform B-Spline Method

Table 11.12 Statistical Summary of Test Set 1 using Non-Uniform B-Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	3.55E-15	7.11E-14	3	1	6.93E-16	1.39E-14	1.24E-15	1.4575
test1b	1.33E-07	2.65E-06	6	2	4.10E-08	8.20E-07	9.33E-08	1.6825
test1c	3.12E-07	1.59E-05	6	1	6.43E-08	3.26E-06	1.50E-07	1.85875
test1d	5.33E-15	1.07E-13	36	2	1.31E-15	2.62E-14	4.73E-09	0.93437
test1e	1.02E-06	2.04E-05	27	8	3.03E-07	6.06E-06	5.70E-07	1.04875
test1f	1.37E-03	6.85E-02	47	4	3.70E-04	1.85E-02	4.51E-04	0.365
test1g	4.44E-15	8.88E-14	3	1	1.15E-15	2.29E-14	2.32E-15	0.32
test1h	1.01E-06	2.03E-05	12	10	2.17E-07	4.33E-06	3.80E-07	0.80062
test1i	1.37E-03	6.85E-02	4	1	3.69E-04	1.85E-02	7.62E-04	0.2825
test1j	4.44E-15	8.88E-14	19	2	1.04E-15	2.09E-14	2.41E-05	0.87125
test1k	9.46E-07	1.89E-05	22	2	2.53E-07	5.06E-06	8.94E-07	1.06094
test1l	1.29E-03	6.52E-02	30	2	3.65E-04	1.84E-02	6.15E-04	1.16625
test1m	3.55E-14	7.11E-14	3	2	1.24E-14	2.49E-14	6.18E-05	2.42031
test1n	2.65E-10	2.60E-08	6	1	8.20E-11	8.04E-09	6.21E-06	2.45031
test1o	2.65E-10	1.33E-06	6	1	8.20E-11	4.10E-07	6.24E-07	2.48922
test1p	1.13E-04	5.66E-03	6	17	4.05E-05	2.03E-03	6.68E-05	0.36844
test1q	9.69E-04	4.85E-02	95	5	3.42E-04	1.71E-02	5.62E-04	1.16672
test1r	4.04E-05	2.02E-03	178	5	1.36E-05	6.78E-04	2.25E-05	9.95125
test1s	1.13E-06	5.64E-03	6	10	4.05E-07	2.03E-03	8.99E-07	0.21062
test1t	9.69E-06	4.85E-02	95	4	3.42E-06	1.71E-02	5.62E-06	0.90078
test1u	4.00E-07	2.00E-03	178	5	1.36E-07	6.77E-04	2.42E-07	9.0818
test1a1	4.16E-17	8.33E-14	3	1	1.03E-17	2.05E-14	2.43E-08	2.90313
test1b1	1.33E-09	2.65E-06	6	1	4.10E-10	8.20E-07	2.62E-09	2.92234
test1c1	7.81E-09	1.59E-05	6	1	1.61E-09	3.26E-06	3.74E-09	2.93172
test1d1	5.55E-17	1.11E-13	36	4	1.32E-17	2.65E-14	1.18E-10	1.84336
test1e1	5.84E-09	1.17E-05	42	2	1.29E-09	2.59E-06	2.27E-09	1.84242
test1f1	1.33E-05	6.65E-02	36	9	4.87E-06	2.44E-02	8.02E-06	1.86543
test1g1	4.86E-17	9.71E-14	4	1	1.44E-17	2.87E-14	2.54E-07	1.72953
test1h1	5.48E-09	1.10E-05	12	1	1.30E-09	2.60E-06	8.33E-09	1.72433
test1i1	1.37E-05	6.85E-02	4	18	3.70E-06	1.85E-02	7.62E-06	1.85921
test1j1	4.86E-17	9.71E-14	20	6	1.18E-17	2.36E-14	2.41E-07	1.91371
test1k1	7.68E-09	1.54E-05	23	2	1.56E-09	3.13E-06	8.06E-09	1.83808
test1l1	1.29E-05	6.51E-02	30	4	3.65E-06	1.84E-02	6.15E-06	1.90277
test1a2	3.55E-15	7.11E-14	3	1	6.93E-16	1.39E-14	1.24E-15	1.47
test1b2	2.67E-15	5.33E-14	3	1	4.60E-16	9.19E-15	7.72E-16	1.6875
test1c2	1.11E-15	5.64E-14	9	8	2.24E-16	1.14E-14	3.57E-16	1.85125
test1d2	5.33E-15	1.07E-13	36	2	1.31E-15	2.62E-14	1.48E-15	0.86937
test1e2	9.37E-07	1.87E-05	2	17	3.88E-07	7.75E-06	6.45E-07	1.10625
test1f2	1.25E-03	6.23E-02	39	19	3.24E-04	1.62E-02	4.53E-04	1.23156
test1g2	4.44E-15	8.88E-14	3	1	1.15E-15	2.29E-14	1.43E-05	1.23469
test1h2	9.45E-07	1.89E-05	12	4	2.51E-07	5.02E-06	6.33E-07	1.20485
test1i2	1.25E-03	6.23E-02	4	2	3.24E-04	1.62E-02	6.80E-04	1.26375
test1j2	4.44E-15	8.88E-14	19	2	1.04E-15	2.09E-14	2.15E-05	1.43344
test1k2	6.27E-07	1.25E-05	29	11	2.64E-07	5.29E-06	8.38E-07	1.45078
test1l2	1.16E-03	5.81E-02	21	12	4.20E-04	2.11E-02	7.19E-04	1.50539
test1m2	3.55E-14	7.11E-14	3	2	1.24E-14	2.49E-14	7.23E-05	2.77219
test1n2	8.88E-16	8.71E-14	1	1	1.98E-16	1.94E-14	7.27E-06	2.79156
test1o2	1.39E-17	6.94E-14	9	9	3.21E-18	1.61E-14	7.30E-07	2.77031
test1p2	1.13E-04	5.66E-03	5	6	4.06E-05	2.03E-03	6.69E-05	0.2243
test1q2	9.69E-04	4.85E-02	5	95	3.42E-04	1.71E-02	5.62E-04	1.05281
test1r2	4.03E-05	2.01E-03	5	178	1.36E-05	6.78E-04	2.42E-05	10.64711
test1s2	1.13E-06	5.64E-03	5	6	4.05E-07	2.03E-03	9.30E-07	0.01609
test1t2	9.69E-06	4.85E-02	5	95	3.42E-06	1.71E-02	5.62E-06	0.85945
test1u2	3.99E-07	1.99E-03	4	178	1.36E-07	6.78E-04	2.42E-07	10.71055
test1a12	4.16E-17	8.33E-14	3	1	1.03E-17	2.05E-14	2.43E-08	3.16539
test1b12	2.78E-17	5.55E-14	3	1	4.82E-18	9.65E-15	2.44E-09	3.20004
test1c12	3.47E-17	7.05E-14	9	3	5.81E-18	1.18E-14	2.45E-10	3.18461
test1d12	5.55E-17	1.11E-13	36	4	1.32E-17	2.65E-14	7.76E-12	2.15067
test1e12	7.11E-09	1.42E-05	38	17	1.36E-09	2.71E-06	2.37E-09	2.01488
test1f12	1.18E-05	5.94E-02	10	15	4.77E-06	2.39E-02	7.88E-06	2.00949

Table 11.12 Statistical Summary of Test Set 1 using Non-Uniform B-Splines (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1g12	4.86E-17	9.71E-14	4	1	1.44E-17	2.87E-14	2.49E-07	1.9321
test1h12	3.78E-09	7.56E-06	6	16	1.40E-09	2.80E-06	8.16E-09	1.92943
test1i12	1.25E-05	6.23E-02	3	2	3.24E-06	1.62E-02	6.81E-06	1.97693
test1j12	4.86E-17	9.71E-14	20	6	1.18E-17	2.36E-14	2.15E-07	2.07437
test1k12	4.98E-09	9.96E-06	19	10	1.50E-09	3.00E-06	7.28E-09	1.99167
test1l12	1.16E-05	5.80E-02	19	12	4.20E-06	2.11E-02	7.19E-06	1.96924
test1aa	5.00E-07	9.69E-06	7	6	1.80E-07	3.49E-06	1.81E-07	-1.54
test1bb	1.82E-05	3.63E-04	6	5	1.51E-06	3.03E-05	1.52E-06	-1.7525
test1cc	5.59E-05	2.73E-03	6	5	4.05E-06	1.98E-04	4.07E-06	-1.91125
test1dd	7.21E-07	1.39E-05	19	9	2.24E-07	4.31E-06	2.58E-07	-1.49875
test1ee	0.2737	5.473	40	1	3.98E-02	0.7955	3.98E-02	-1.71719
test1ff	0.3597	17.17	33	1	4.99E-02	2.384	5.00E-02	-1.84437
test1gg	7.21E-07	1.39E-05	32	12	2.24E-07	4.31E-06	1.58E-03	-1.81312
test1hh	0.2529	5.058	17	2	4.48E-03	8.96E-02	4.49E-03	-1.89609
test1ii	0.4766	22.99	7	1	6.54E-03	0.3154	6.54E-03	-1.92531
test1jj	6.52E-07	1.26E-05	17	11	2.34E-07	4.54E-06	2.07E-04	-2.05531
test1kk	0.2544	5.087	31	10	2.27E-02	0.4539	2.27E-02	-2.11422
test1ll	0.3883	18.71	18	1	2.91E-02	1.4	2.91E-02	-2.1518
test1mm	4.70E-06	9.37E-06	6	5	1.47E-06	2.94E-06	2.92E-03	-2.83812
test1nn	8.29E-07	7.06E-05	5	9	2.33E-07	1.99E-05	2.94E-04	-2.83727
test1oo	8.61E-08	4.96E-05	4	5	8.26E-09	4.76E-06	2.95E-05	-2.85617
test1pp	0.5396	25.04	70	13	4.49E-02	2.082	4.49E-02	-1.00055
test1qq	0.6238	28.99	100	13	6.85E-02	3.184	6.86E-02	-0.38961
test1rr	0.2826	13.04	15	9	3.88E-02	1.789	3.88E-02	6.60398
test1ss	5.40E-03	2.848	70	13	4.49E-04	0.2367	1.34E-03	-1.15113
test1tt	6.24E-03	3.213	100	13	6.85E-04	0.3529	6.86E-04	-0.56676
test1uu	2.12E-03	1.073	5	9	3.52E-04	0.1778	3.54E-04	-0.66933
test1bb2	1.33E-07	2.65E-06	6	2	4.10E-08	8.20E-07	9.33E-08	1.54
test1cc2	3.12E-07	1.59E-05	6	1	6.43E-08	3.26E-06	1.50E-07	1.68
test1dd2	5.33E-15	1.07E-13	36	2	1.31E-15	2.62E-14	4.73E-09	0.8475
test1ee2	1.02E-06	2.04E-05	27	8	3.03E-07	6.06E-06	5.70E-07	1.10125
test1ff2	1.37E-03	6.85E-02	47	4	3.70E-04	1.85E-02	4.51E-04	1.16625
test1gg2	4.44E-15	8.88E-14	3	1	1.15E-15	2.29E-14	1.43E-05	1.26094
test1hh2	1.01E-06	2.03E-05	12	10	2.17E-07	4.33E-06	5.90E-07	1.20766
test1ii2	1.37E-03	6.85E-02	4	1	3.69E-04	1.85E-02	7.62E-04	1.32641
test1jj2	4.44E-15	8.88E-14	19	2	1.04E-15	2.09E-14	2.41E-05	1.50453
test1kk2	9.46E-07	1.89E-05	22	2	2.53E-07	5.06E-06	8.94E-07	1.56016
test1ll2	1.29E-03	6.52E-02	30	2	3.65E-04	1.84E-02	6.15E-04	1.51313
test1mm2	3.55E-14	7.11E-14	3	2	1.24E-14	2.49E-14	6.18E-05	2.76961
test1nn2	2.65E-10	2.60E-08	6	1	8.20E-11	8.04E-09	6.21E-06	2.77867
test1oo2	2.65E-10	1.33E-06	6	1	8.20E-11	4.10E-07	6.24E-07	2.77781
test1pp2	1.13E-04	5.66E-03	6	17	4.05E-05	2.03E-03	6.68E-05	0.11336
test1qq2	9.69E-04	4.85E-02	95	5	3.42E-04	1.71E-02	5.62E-04	0.72742
test1rr2	4.04E-05	2.02E-03	178	5	1.36E-05	6.78E-04	2.42E-05	8.94578
test1ss2	1.13E-06	5.64E-03	6	10	4.05E-07	2.03E-03	9.30E-07	0.30723
test1tt2	9.69E-06	4.85E-02	95	4	3.42E-06	1.71E-02	5.62E-06	0.66813
test1uu2	3.73E-07	1.87E-03	11	6	1.26E-07	6.34E-04	3.57E-07	0.4841
test1A	1.33E-07	1.33E-06	6	3	4.10E-08	4.10E-07	4.12E-08	-1.5375
test1B	1.33E-07	1.90E-06	6	4	4.10E-08	5.88E-07	4.14E-08	-1.75625
test1C	1.48E-06	1.48E-05	17	3	3.94E-07	3.94E-06	3.95E-07	-1.3125
test1D	1.57E-06	1.57E-05	29	19	3.64E-07	3.64E-06	3.64E-07	-1.5925
test1E	1.30E-03	1.85E-02	30	3	3.65E-04	5.23E-03	3.66E-04	-1.77437
test1F	1.16E-03	1.66E-02	10	12	4.20E-04	6.01E-03	4.20E-04	-1.84813
test1G	1.07E-06	5.26E-05	11	10	3.12E-07	1.53E-05	1.33E-05	-1.93719
test1H	1.16E-03	3.84E-02	48	9	4.20E-04	1.39E-02	4.20E-04	-2.03297
test1I	1.47E-03	3.68E-02	20	9	4.93E-04	1.24E-02	4.93E-04	-0.61594
test1J	7.87E-04	3.90E-02	61	8	3.43E-04	1.70E-02	3.43E-04	21.69789
test1A1	1.86E-05	1.86E-04	5	6	1.65E-06	1.65E-05	3.45E-05	-3.0616
test1B1	1.86E-05	2.67E-04	5	5	1.61E-06	2.31E-05	3.82E-06	-3.06625
test1C1	0.2544	2.544	31	10	2.27E-02	0.2269	2.27E-02	-2.43234

Table 11.12 Statistical Summary of Test Set 1 using Non-Uniform B-Splines (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1D1	0.2544	2.544	31	10	2.27E-02	0.2269	2.27E-02	-2.44664
test1E1	0.3883	5.558	18	1	2.91E-02	0.4158	2.91E-02	-2.4907
test1F1	0.2537	3.627	31	10	2.30E-02	0.3281	2.30E-02	-2.47461
test1G1	5.09E-04	2.33E-02	31	10	4.56E-05	2.08E-03	7.29E-04	-2.48911
test1H1	1.67E-03	5.36E-02	31	9	4.65E-04	1.50E-02	4.66E-04	-2.50317
test1I1	0.6267	15.37	39	1	5.15E-02	1.262	5.15E-02	-1.07929
test1J1	0.5981	27.44	8	1	5.79E-02	2.656	5.79E-02	21.47398
test1A2	1.10E-06	1.10E-05	2	1	3.80E-07	3.80E-06	5.82E-03	-3.31212
test1B2	1.10E-06	1.54E-05	2	1	2.91E-07	4.07E-06	5.85E-04	-3.30441
test1C2	5.15E-06	5.06E-05	3	7	6.34E-07	6.22E-06	1.85E-05	-2.57732
test1D2	1.61E-06	1.61E-05	29	3	4.02E-07	4.02E-06	7.10E-07	-2.56465
test1E2	1.30E-03	1.81E-02	30	6	3.65E-04	5.12E-03	3.65E-04	-2.57217
test1F2	1.16E-03	1.62E-02	41	12	4.20E-04	5.88E-03	4.20E-04	-2.58957
test1G2	1.04E-06	4.27E-05	17	10	3.12E-07	1.29E-05	1.33E-05	-2.58713
test1H2	1.16E-03	3.45E-02	45	9	4.20E-04	1.25E-02	4.20E-04	-2.71475
test1I2	1.47E-03	3.39E-02	20	9	4.93E-04	1.14E-02	4.93E-04	-1.05263
test1J2	7.88E-04	3.27E-02	61	10	3.43E-04	1.42E-02	3.43E-04	21.94878

Table 11.13 Statistical Summary of Test Set 2 using Non-Uniform B-Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test2C	9.45E-02	4.727	3	4	4.83E-03	0.2414	1.60E-02	1.92875
test2D	0.3997	6.662	3	5	3.34E-02	0.5569	7.94E-02	1.99125
test2E	0.4338	22.02	3	5	0.1112	5.648	0.2073	2.11875
test2F	1.78E-15	8.88E-14	21	3	4.23E-16	2.11E-14	4.15E-03	0.75063
test2G	0.2131	10.65	16	25	3.06E-03	0.1529	8.64E-03	0.3875
test2H	0.1045	5.224	20	23	1.03E-03	5.13E-02	6.66E-03	0.30844
test2I	1.092	18.19	16	25	1.52E-02	0.2536	4.22E-02	0.17281
test2J	0.8322	41.9	16	25	6.58E-02	3.31	0.1293	0.0518
test2K	1.04E-17	1.04E-13	25	28	2.38E-18	2.38E-14	2.59E-03	0.04875
test2L	1.20E-03	11.97	16	25	1.66E-05	0.1659	7.11E-05	0.06359
test2M	9.34E-04	9.343	16	25	1.78E-05	0.1783	5.91E-05	0.04601
test2N	8.32E-04	41.9	16	25	6.58E-05	3.31	1.29E-04	0.03758
test2A1	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	-1.625
test2B1	8.75E-02	4.377	3	5	6.86E-03	0.343	6.90E-03	-1.8125
test2C1	9.45E-02	4.727	3	4	4.83E-03	0.2414	4.90E-03	-1.95125
test2D1	0.3997	6.662	3	5	3.34E-02	0.5569	3.36E-02	-2.06625
test2E1	0.4338	21.59	3	5	0.1112	5.537	0.1118	-2.15062
test2F1	2	91.81	8	21	0.112	5.141	0.112	-0.59625
test2G1	1.894	94.68	12	16	0.1001	5.005	0.1001	-0.87813
test2H1	1.999	99.94	2	19	0.1114	5.572	0.1115	-0.98531
test2I1	5.467	91.12	2	16	0.2748	4.58	0.2749	-1.35438
test2L1	9.46E-03	5.151	2	16	4.94E-04	0.2688	5.52E-03	-1.27164
test2M1	7.36E-03	4.004	2	25	3.64E-04	0.1979	3.80E-04	-1.28328
test2N1	1.51E-03	0.8445	15	21	1.13E-04	6.34E-02	1.15E-04	-1.50594

Table 11.14 Statistical Summary of Test Set 3 using Non-Uniform B-Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A	1.34E-03	2.68E-02	1	2	1.16E-04	2.32E-03	4.12E-04	1.475
test3B	3.61E-02	1.035	2	7	4.82E-03	0.1383	1.18E-02	1.71875
test3C	1.79E-04	3.59E-03	1	1	3.89E-05	7.78E-04	3.78E-04	0.71375
test3D	1.34E-03	2.68E-02	1	4	2.21E-04	4.43E-03	5.87E-04	0.93375
test3E	3.78E-02	1.081	31	7	5.46E-03	0.1562	9.96E-03	1.12344
test3F	4.61E-02	1.317	18	12	4.85E-03	0.1386	1.35E-02	1.1875
test3G	3.60E-03	0.3528	1	9	6.73E-04	6.60E-02	1.81E-03	1.30141
test3A	1.34E-03	2.68E-02	1	2	1.16E-04	2.32E-03	4.12E-04	1.44
test3B	3.61E-02	1.035	2	7	4.82E-03	0.1383	1.18E-02	1.675
test3C	1.79E-04	3.59E-03	1	1	3.89E-05	7.78E-04	3.78E-04	0.64375
test3D	1.34E-03	2.68E-02	1	4	2.21E-04	4.43E-03	5.87E-04	0.91188
test3E	3.78E-02	1.081	31	7	5.46E-03	0.1562	9.96E-03	1.01
test3F	4.61E-02	1.317	18	12	4.85E-03	0.1386	1.35E-02	1.27906
test3G	3.60E-03	0.3528	1	9	6.73E-04	6.60E-02	1.81E-03	1.24688
test3A1	4.167	83.33	3	2	0.1591	3.182	0.1599	-1.4975
test3B1	2.929	84.06	3	3	0.1194	3.427	0.1211	-1.73875
test3A1	4.186	83.73	3	2	0.1852	3.704	0.746	1.425
test3B1	2.969	85.19	3	3	0.1439	4.129	0.5357	1.66875
test3C1	4.508	90.15	20	10	3.50E-02	0.6994	0.3432	0.5725
test3D1	4.082	81.65	20	8	3.08E-02	0.6169	0.2905	0.9625
test3E1	3.148	90.13	21	8	3.13E-02	0.8966	0.2225	1.00875
test3F1	2.977	85.13	20	9	2.79E-02	0.7972	0.2109	1.22969
test3G1	1.103	94.06	21	10	1.60E-02	1.365	8.51E-02	1.19859
test3H1	1.56	97.26	21	9	2.08E-02	1.298	0.1067	1.38875
test3A3	1.55E-15	7.77E-14	2	3	2.35E-16	1.18E-14	3.76E-16	1.56
test3B3	0.2239	11.2	10	6	1.76E-02	0.8778	4.46E-02	1.765
test3C3	0.1175	5.877	3	1	9.63E-03	0.4816	2.46E-02	1.89375
test3D3	1.12	18.66	10	6	9.45E-02	1.575	0.2241	2.04375
test3E3	1.409	71.52	10	10	0.2386	12.12	0.4825	2.10188
test3F3	1.78E-15	8.88E-14	15	19	7.75E-17	3.87E-15	9.65E-03	0.19937
test3G3	0.3398	16.99	16	25	1.12E-02	0.5573	3.32E-02	0.04187
test3H3	0.1233	6.167	19	1	7.78E-03	0.3888	2.06E-02	0.19906
test3I3	1.772	29.53	16	25	6.28E-02	1.046	0.1666	0.28766
test3J3	1.409	70.92	50	50	0.3354	16.89	0.6642	0.53805
test3K3	1.04E-17	1.04E-13	30	25	4.80E-19	4.80E-15	1.33E-02	0.48211
test3L3	2.06E-03	20.62	16	25	9.25E-05	0.9253	3.28E-04	0.54695
test3M3	1.34E-03	13.35	16	25	9.19E-05	0.919	2.27E-04	0.58172
test3N3	1.41E-03	70.92	50	50	3.35E-04	16.89	6.64E-04	0.78711

Table 11.15 Statistical Summary of Test Set 4 using Non-Uniform B-Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test4A2	1.23E-06	2.45E-05	16	15	3.37E-07	6.74E-06	5.45E-07	0.39
test4A3	1.46E-06	2.92E-05	31	28	3.41E-07	6.81E-06	5.34E-07	6.105
test4B2	5.83E-02	1.676	9	29	2.45E-02	0.7041	3.91E-02	0.39594
test4B3	5.87E-02	1.687	18	58	2.50E-02	0.7183	3.95E-02	5.88109
test4C2	9.65E-07	1.93E-05	154	53	3.13E-07	6.26E-06	3.13E-04	19.2761
test4C3	1.05E-06	2.11E-05	317	15	3.26E-07	6.52E-06	1.33E-06	82.79489
test4D2	1.36E-06	2.72E-05	153	6	3.20E-07	6.40E-06	5.02E-07	18.3591
test4D3	1.46E-06	2.91E-05	318	78	3.23E-07	6.46E-06	5.00E-07	83.02826
test4E2	5.87E-02	1.687	188	6	1.77E-02	0.5093	3.58E-02	18.06747
test4E3	5.87E-02	1.685	305	58	1.78E-02	0.5108	3.59E-02	82.10675
test4F2	5.66E-02	1.626	103	35	1.46E-02	0.4198	2.59E-02	17.95575
test4F3	5.85E-02	1.682	251	69	1.48E-02	0.4251	2.61E-02	83.36497
test4G2	1.04E-06	1.02E-04	160	48	2.95E-07	2.89E-05	2.07E-04	18.81665
test4H2	5.66E-02	3.795	167	35	1.46E-02	0.9797	2.59E-02	19.1125
test4H3	5.85E-02	3.933	357	69	1.48E-02	0.9941	2.61E-02	85.93512
test4A12	5	100	1	1	0.2886	5.772	0.9982	0.385
test4A13	5	100	1	1	0.2654	5.307	0.9486	5.66563
test4B12	3.476	100	1	10	0.2309	6.644	0.6993	0.74891
test4B13	3.479	100	1	19	0.2168	6.232	0.6681	5.40453
test4C12	5	100	1	1	1.432	28.64	2.231	15.44758
test4C13	5	100	1	1	1.414	28.28	2.222	70.74551
test4D12	5	100	1	1	1.364	27.29	2.13	15.17609
test4D13	5	100	1	1	1.35	27	2.125	70.38852
test4E12	3.13	89.94	118	1	0.7312	21.01	1.128	15.07135
test4E13	3.135	90	236	1	0.7249	20.81	1.127	70.47375
test4F12	3.482	100	1	33	0.8174	23.48	1.26	14.55573
test4F13	3.477	100	1	65	0.8083	23.25	1.256	68.94855
test4G12	1.051	87.21	118	39	0.3012	25	0.453	14.14227
test4A12	5	100	1	1	0.2886	5.772	0.9982	0.42
test4A22	0.2848	5.696	21	21	7.12E-02	1.424	6.65E-02	0.5675
test4A23	0.2925	5.849	40	40	7.31E-02	1.462	6.64E-02	6.40188
test4B22	0.2864	8.24	20	21	7.57E-02	2.177	7.33E-02	0.62359
test4B23	0.2933	8.43	41	41	7.74E-02	2.224	7.32E-02	5.84641
test4C22	0.19	3.8	174	40	1.05E-02	0.2107	2.36E-02	19.21438
test4C23	0.1947	3.893	349	81	1.07E-02	0.213	2.37E-02	82.5634
test4D22	0.19	3.8	174	40	1.05E-02	0.2107	2.36E-02	17.8689
test4D23	0.1947	3.893	349	81	1.07E-02	0.213	2.37E-02	83.12866
test4E22	0.1895	5.444	175	41	1.94E-02	0.556	3.73E-02	17.53806
test4E23	0.1951	5.601	349	80	1.94E-02	0.5581	3.74E-02	81.88776
test4F22	0.1942	5.577	174	43	1.95E-02	0.5612	3.31E-02	17.83801
test4F23	0.196	5.636	349	81	1.98E-02	0.5685	3.33E-02	82.63739
test4G22	0.19	15.77	174	40	1.05E-02	0.8743	2.36E-02	16.79657
test4G23	0.1947	16.09	349	80	1.07E-02	0.8804	2.37E-02	82.75874
test4H22	0.1942	12.3	174	43	1.95E-02	1.238	3.31E-02	18.31856
test4H23	0.196	12.46	349	81	1.98E-02	1.257	3.33E-02	82.68677

11.4 Thin-Plate Spline Method

The Thin-Plate Spline (TPS) method is a hybrid of the Multiquadrics and the Infinite-Plate Spline methods. Indeed, it is merely a local version of the latter, generalized to higher dimensionality, and its equations are identical to those of the former except for the basis function used.

The implementation of TPS is described in Chapter 7. Similar to MQ, some options were explored. Among them, the issue of scaling the data to a unit domain or using the data as given, and a global implementation versus local (i.e., domain decomposition or subdomaining). These options were investigated using a partial set of test set 1 to determine the most efficient, yet accurate method of implementing TPS.

Overall Accuracy – Statistical summaries of the TPS results are presented in Tables 11.16 – 11.21. The overall accuracy of the TPS method is very good. The error does not vary dramatically across the range of test case functions; is usually less than 5% and for several functions << 1%. The largest errors occurred in the sinusoidal cases with high number of cycles. Figures 11-30 and 11-31 plot the error for each category of function test cases. It is directly observed from these plots that the method has excellent performance for constant, linear, and for one-cycle sinusoidal functions. There is, however, a tendency for larger errors for the higher-frequency sinusoidal functions, even though the relative error remains less than 10%. For the majority of the runs, there is no large difference between the maximum and the average errors, indicating that the method is not sensitive to any particular location on the surface.

Grid Spacing Sensitivity – Figures 11-30 and 11-31 indicate that the sensitivity of TPS to grid spacing is associated with the function to be interpolated. The sets of runs marked "A" on the figure indicate that the function was transferred to an identical grid. For the sets of data marked with "B", the function was transferred to a grid which was clustered in near the edges – as found in CFD grids. The percentage error increased about 8 orders of magnitude between these two types of grids. However, these errors are less than 2% of the maximum function amplitude or magnitude (most are less than 1×10^{-5}%). One concludes that there is a relatively low sensitivity to the grid spacing.

As an example of a characteristic oscillation in the interpolation result, Figure 11-32 shows a typical interpolation error plot by the TPS method. Figures 11-32 a) and b) show the reference and the interpolated one-cycle sinusoidal function, respectively. In Figures 11-32 c) and d), very small errors concentrate on the border of the domain that vary with the function shape.

Directional Bias – Figure 11-30 includes the streamwise (test1a – 11) and spanwise (test1a2 – 12) functions examined in test set 1. The errors are virtually identical for both function directions, indicating little or no sensitivity.

Magnitude/Amplitude Sensitivity – The TPS method appears to have no relative sensitivity to different amplitudes or magnitudes, as shown in Figure 11-33. This trend is consistent for function type, plates, shells, direction of the function and grid (test sets 1 and 2). This trend indicates that the percentage error (refer to Table 11.16) does not change.

Sensitivity to Frequency (Higher Oscillations) – The TPS method is moderately sensitive to the frequency of the function. As one can see from the results in Tables 11.16 – 19, the errors are very low for constant and linearly varying functions. Figures 11-30 and 11-31 also indicate this trend. There is a large jump in error when a sinusoidally varying function is introduced. The

sinusoidal functions shown in Figures 11-30 and 11-31 have one cycle over the surface. When the frequency of the function is increased, the error increases rapidly. Figure 11-34 shows the results for the test set 1. The percentage error increases from 5 to 11 orders of magnitude as the frequency is increased to three cycles. The large variation in error does not occur as the frequency is increased above three cycles. At this point, the relative error has increased to approximately 20% of the maximum amplitude.

Figure 11-35 illustrates the error present in a high-frequency mode interpolation. Here, the interpolation result for a three-cycle sinusoid is plotted. Notice that while the function amplitude is captured along the surface edges, the amplitude prediction has slightly degraded along the interior of the grid. The oscillations discussed earlier, and plotted for a one-cycle sinusoidal oscillation in Figure 11-32 are now more apparent. These oscillations occur in both directions, showing a close coupling of the error in both of them. These higher-order oscillations illustrate the impact of interpolating higher-order mode shapes as well as pressure distributions.

<u>Extrapolation</u> – The maximum percentage errors for test sets 1 (interpolation) and 3 (extrapolation) are plotted in Figure 11-36. TPS preserves its performance for extrapolations as seen by the constant errors in Figure 11-36. There is basically no difference between the interpolation (represented by "A") and extrapolation (represented by "B") for the linear functions. Sinusoidal functions have the highest overall errors, which is consistent with its results for interpolation.

<u>Diminishing Variation</u> – Figure 11-37 shows that the interpolation errors do not vary with increasing grid fineness, indicating that the TPS scheme is consistent for each function.

<u>Sensitivity to Three-Dimensional Surfaces (Plates Vs. Shells)</u> – From Figures 11-30 to 11-37, the only sensitivity that TPS has when it is extended from two to three dimensions is for the constant and linear functions. The sensitivities due to sinusoidal functions do not change to any significant degree.

<u>Sensitivity to Grid Irregularities</u> – For the irregular grid test set (test set 2), the TPS method performed very well for the constant and linearly varying functions (<<0.01%). There is a large jump in the error when it comes to the sinusoidal functions (~20%), as one can see from Table 11.17. Franke [11] has indicated that the number of elements which define the original grid can be a factor in the accuracy of such a method. However, considering the few points that define the function, the results are still very satisfactory. The grid here is very sparse, and it seems likely that this causes the method to amplify the error associated with those rapidly varying functions. Therefore, it is important that this method be used with grids that have enough points to adequately identify the function shape.

<u>Sensitivity to Subdomaining and Overlapping</u> – The TPS and IPS algorithms are almost identical. The difference in the two methods lies in their method of application. IPS applies the interpolation over the entire surface, resulting in a very large CPU memory requirement. TPS is implemented using a local subdomaining, similar to that implemented for MQ. The term local means that the surface is subdivided into a prescribed number of subdomains. The points within the subdomain are influenced only by the other points within the subdomain. The advantage of subdomaining is that the dimensions of the arrays within the computer code can be reduced, thus reducing the overall memory requirements of the code. In addition, rapidly varying functions can be more accurately interpolated by limiting the zone of influence. A disadvantage of the subdomain approach is that two additional parameters in the form of the subdomain divisions and overlap regions are introduced.

The implementation of the subdomain concept was done based on the maximum number of input points in each direction (x, y, and z) allowed in a given region. This approximately defines the size of the local linear system to be solved. More points enter in the region through the overlapping areas (which was varied between 10% to 30% of the dimension of the subdomain in each direction). Most cases were run using 20 as the maximum number of points in each direction and 10% overlapping. This gives a reasonable number of sub-problems to be solved, and samples a good portion of the original problem. Reducing the number of points in either the subdomains or the overlap regions would increase the performance of the code, but can reduce the accuracy of the interpolation if an insufficient number of points lie in one or more subdomains.

In order to illustrate the change in the accuracy of the method as function of the subdomaining and scaling, test 1p was used with a fixed 10% overlapping in each direction. Since its input grid is a 50×10, subdivisions were made only along the x-direction, where more points were available. Figure 11-38 shows the error of the solution as function of the number of subdomains. Two sets of data were included to show the effect of scaling each subdomain to a unit square. It is apparent that there is no significant difference in the error due to these two parameters.

Sensitivity to Planar Curves (Beam-Like Cases) – Tables 11.20 and 11.21 summarize the statistical results obtained from TPS when interpolating data along a planar curve. The method performed very well, with most errors $<<$ 1%. Due to the formulation of the method, the minimum dimension of the data set must be taken into account when posing the set of equations to be solved. This reduces the number of constraints accordingly. For the set of test cases studied, there were no significant differences obtained by scaling the original data.

Algorithm CPU Memory and Time Requirements – The average CPU time requirement is approximately 1–3 seconds (refer to the last column of Tables 11.16 – 11.21), with the exception of the test cases that had very large errors, indicative of ill-conditioned matrices for the higher-frequency sinusoidal functions. These runs required on the average of 200 to 400 seconds. All the cases were based on a maximum of 20 input data point in each direction of the subdomain and a 10% overlapping of the subregions.

The algorithm CPU memory requirement can be determined using the following algorithm:

$$\text{CPU SIZE (mega bytes)} = 0.3 + 9.1374 \times 10^{-4}\,\text{KGS} + 1.0839 \times 10^{-7}\,\text{KGS}^2 + 3.0867 \times 10^{-4}\,\text{UKS} + 8.0995 \times 10^{-11}\,\text{UKS}^2$$

where KGS is the total number of points for the grid where the function is known and UKS is the total number of mesh points for the grid to which the function is to be interpolated. For example, if mode shapes are to be interpolated from a structural mesh to a CFD mesh, KGS is the number of mesh points for the structural mesh, and UKS is the number of surface CFD grid points.

The influence of the subdomaining and the size of the overlapping in the memory requirement is shown in Figure 11-40. The data is based on 150×150 input grid and 200×200 output grid. There is a drastic reduction in the amount of memory required when the subdomaining technique is used, which is reflected directly in the CPU time. Figure 11-41 shows the actual memory requirements and CPU time for the test 1p used to study the accuracy of subdomaining. It is apparent that the main advantages of subdomaining is the decrease in the overall CPU time and memory requirements.

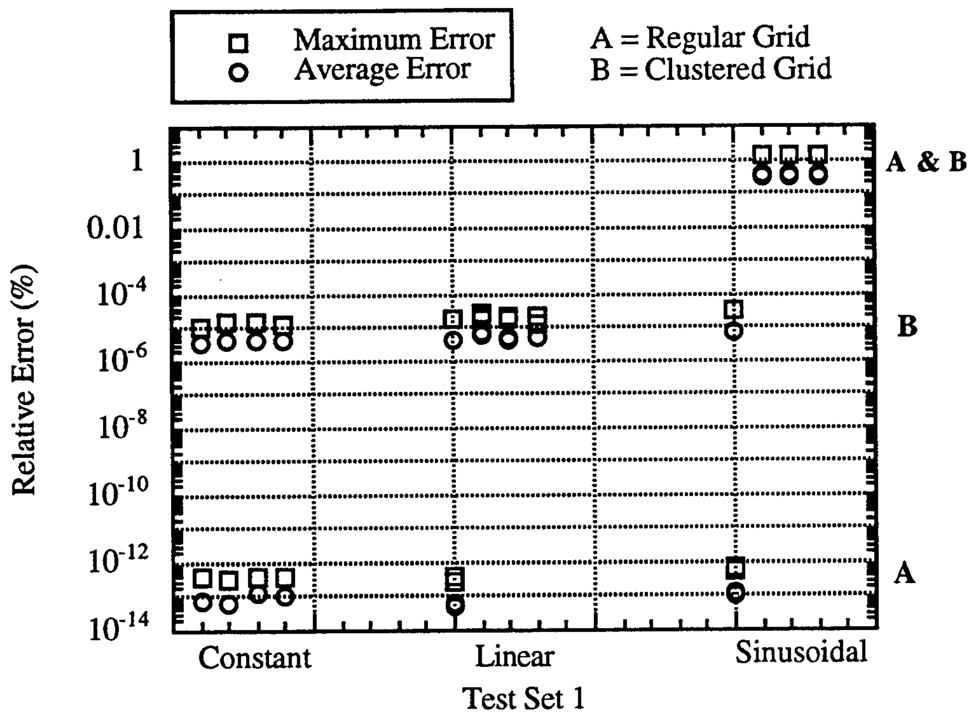


Figure 11-30. Variation of Error for Test Set One Based Upon Function Type for Plates Using Thin-Plate Splines

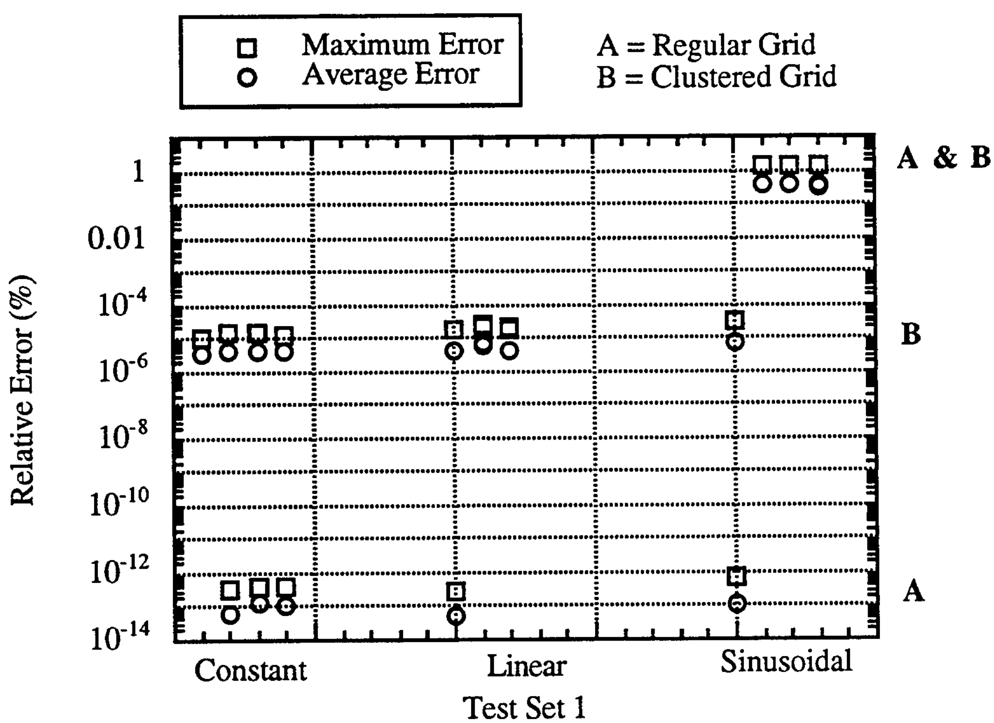
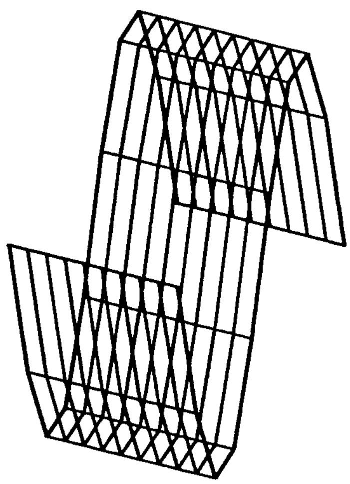
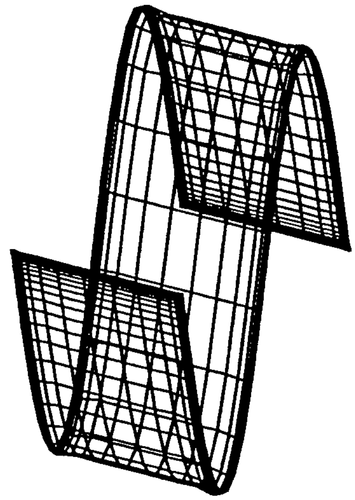


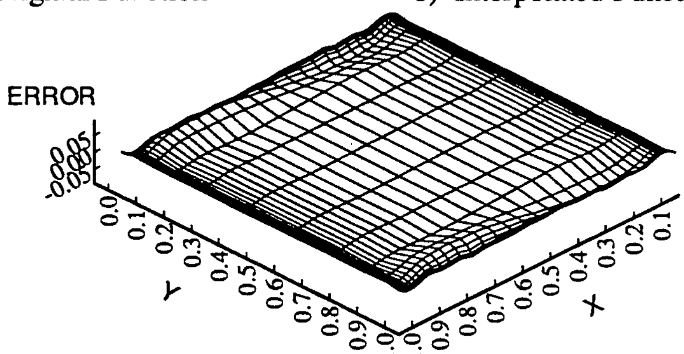
Figure 11-31. Variation of Error for Test Set One Based Upon Function Type for Shells Using Thin-Plate Splines



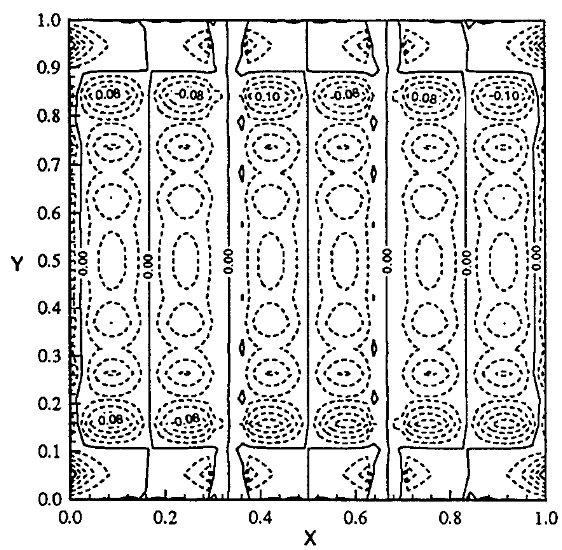
a) Original Function



b) Interpolated Function



c) Orthogonal View of the Error



d) Error Contours

Figure 11-32. Example of Oscillations Induced by the Thin-Plate Spline Method (Test 11) for a One-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2

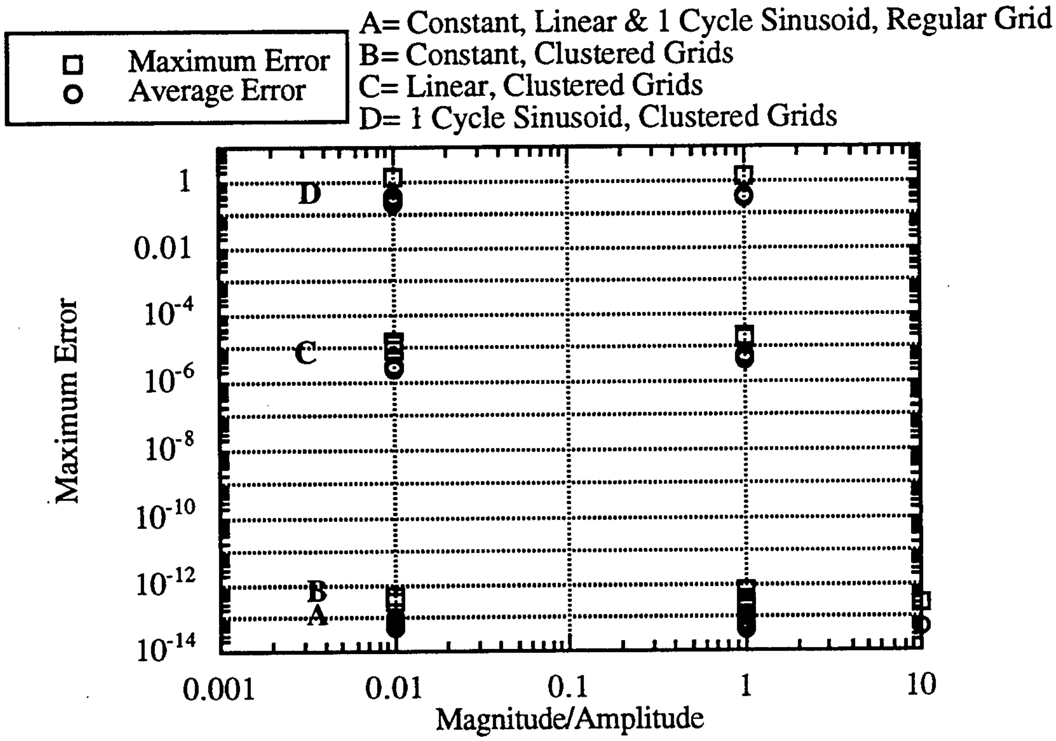


Figure 11-33. Variation of Error for Test Set One Based Upon the Order of Magnitude of the Function Using Thin-Plate Splines

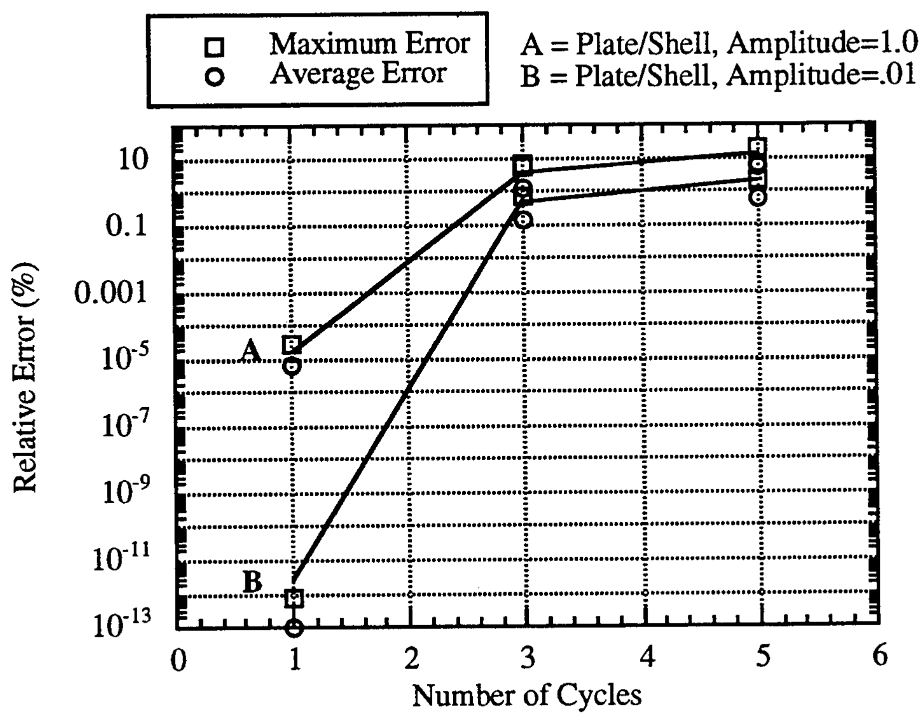


Figure 11-34. Variation of Error for Test Set One Based Upon the Number of Sinusoidal Cycles on the Surface Using Thin-Plate Splines

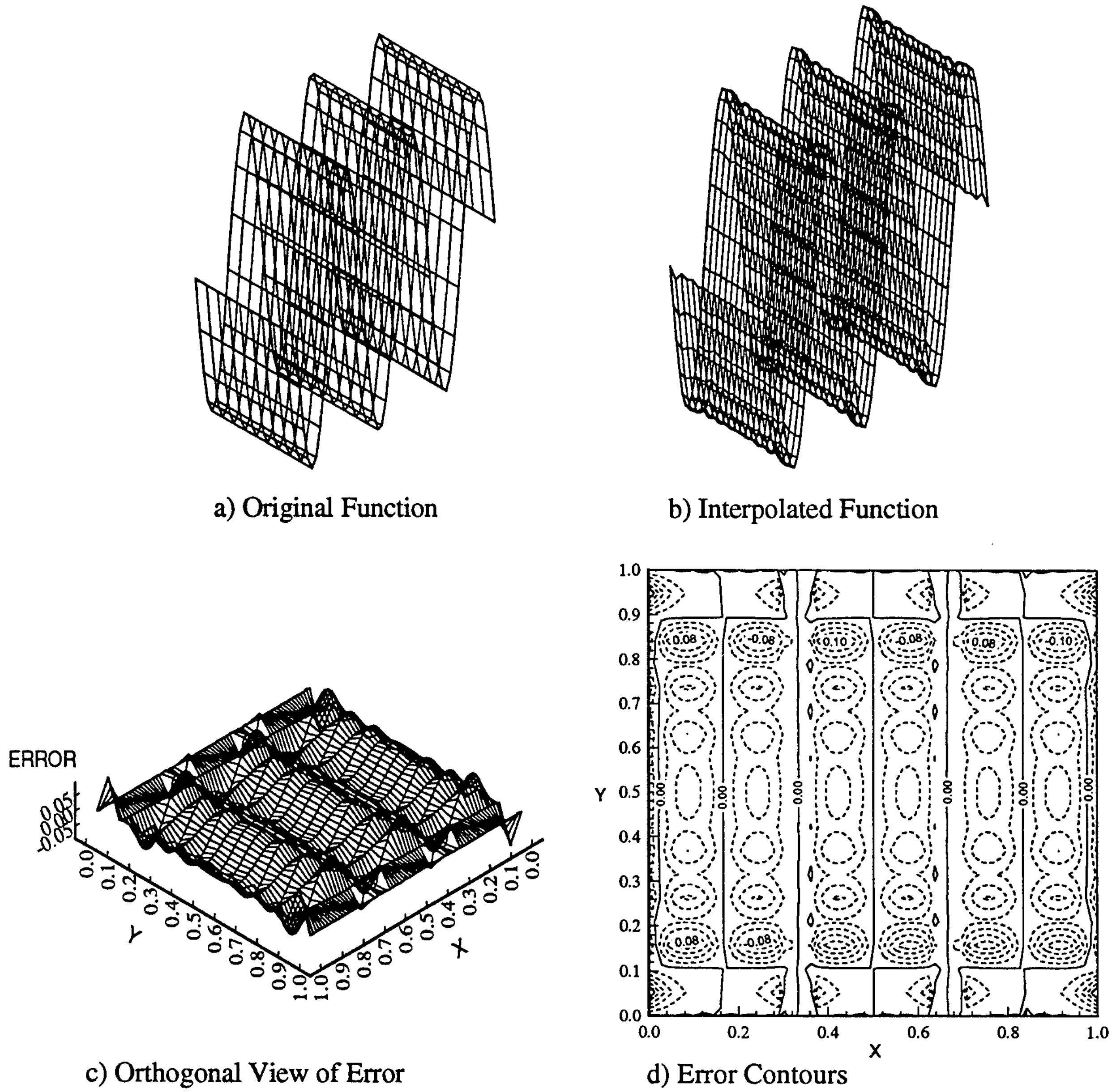


Figure 11-35. Example of Oscillations Induced by the Thin-Plate Spline Method (Test 1p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2 (X-axis and Y-axis have been expanded for visibility)

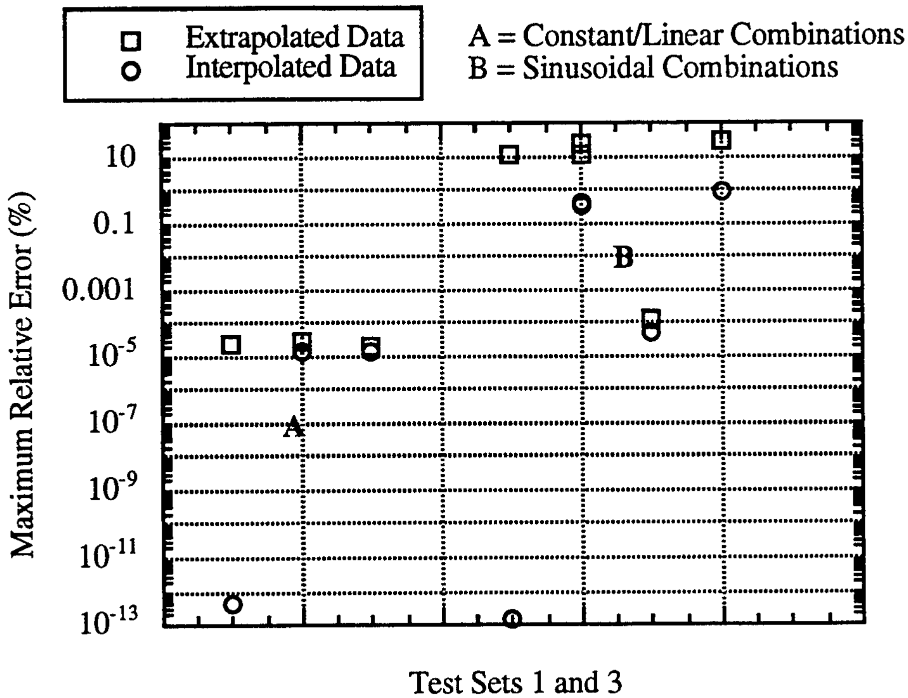


Figure 11-36. Variation of Error for Interpolation (Test Set 1) and Extrapolation (Test Set 3) Using Thin-Plate Splines

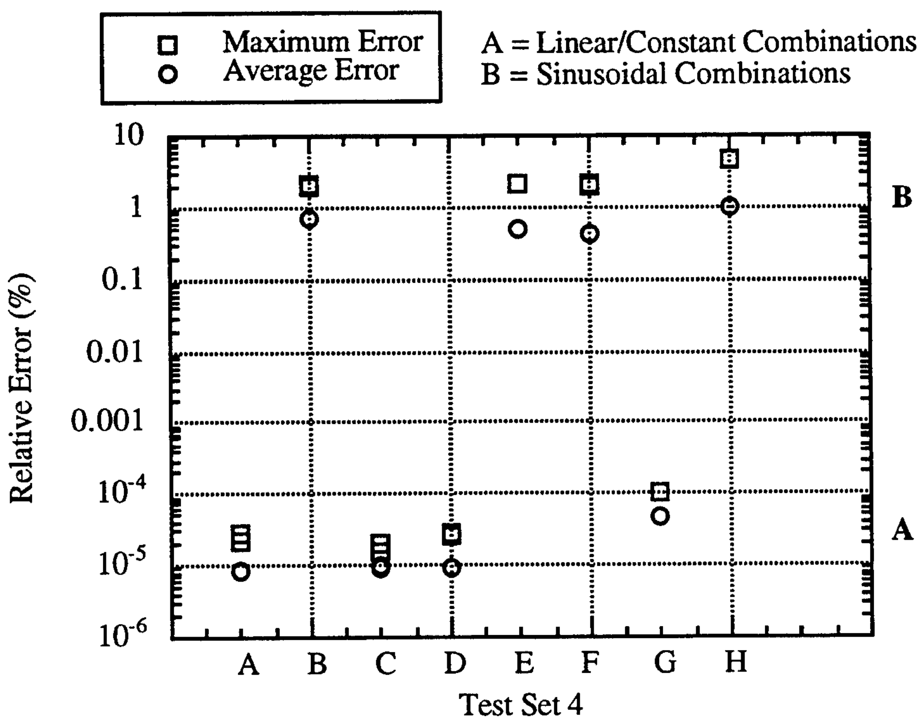


Figure 11-37. Variation of Error for Increasing Grid Fineness (Test Set 4) Using Thin-Plate Splines

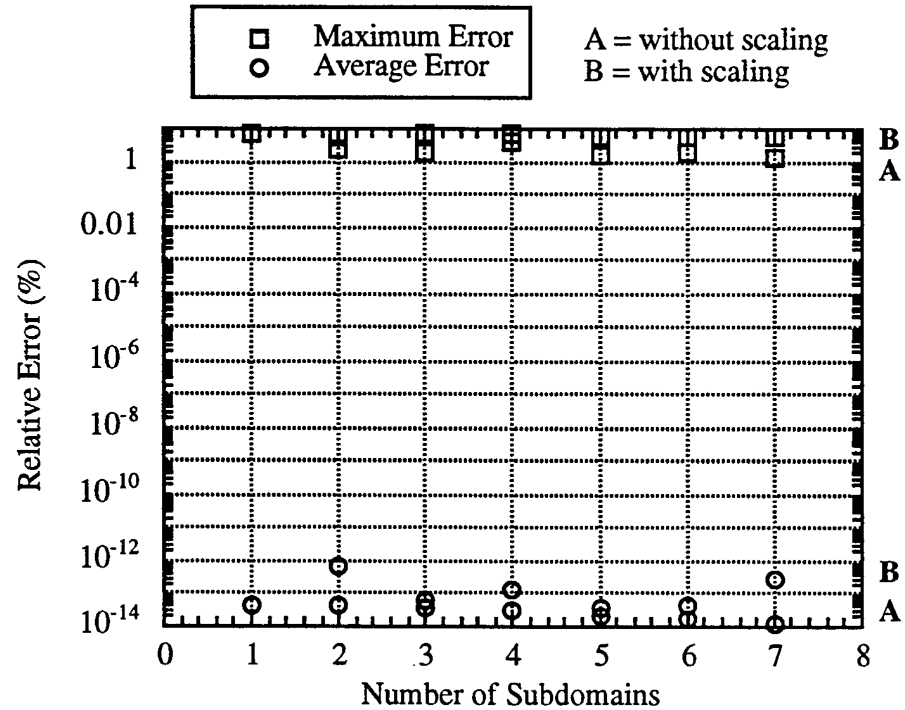


Figure 11-38. Variation of the Maximum Error as Functions of the Number of Subdomains and the Presence or Absence of Scaling for the Thin-Plate Spline Method (Test 1p)

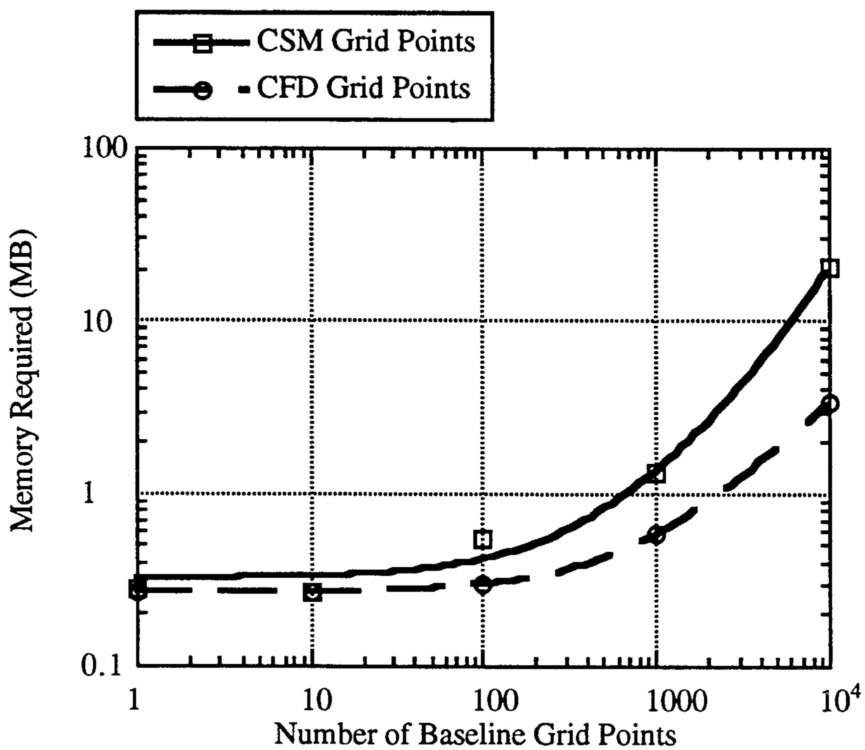


Figure 11-39. CPU Memory Requirements for the Thin-Plate Spline Method as Implemented with 10% Overlapping and No More than 20 Points Along Each Direction of the Subdomain

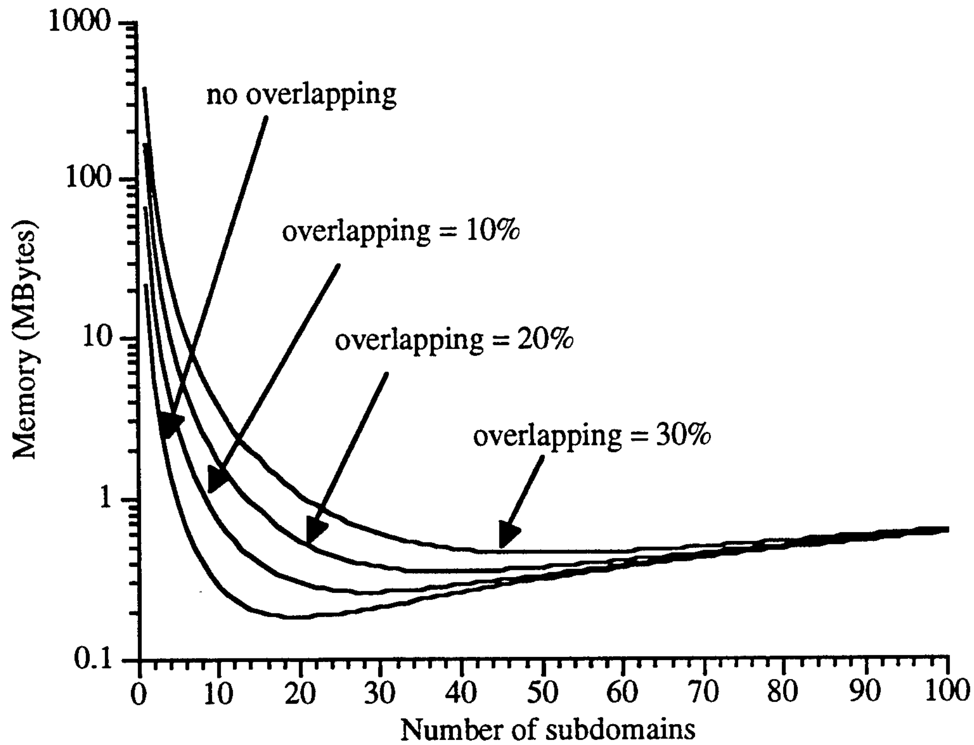


Figure 11-40. Influence of the Size of the Overlapping Region and the Subdomaining on the Memory Requirement for a Hypothetical Test Case (input grid = 150 x 150; output grid = 200 × 200) for the Thin-Plate Spline Method

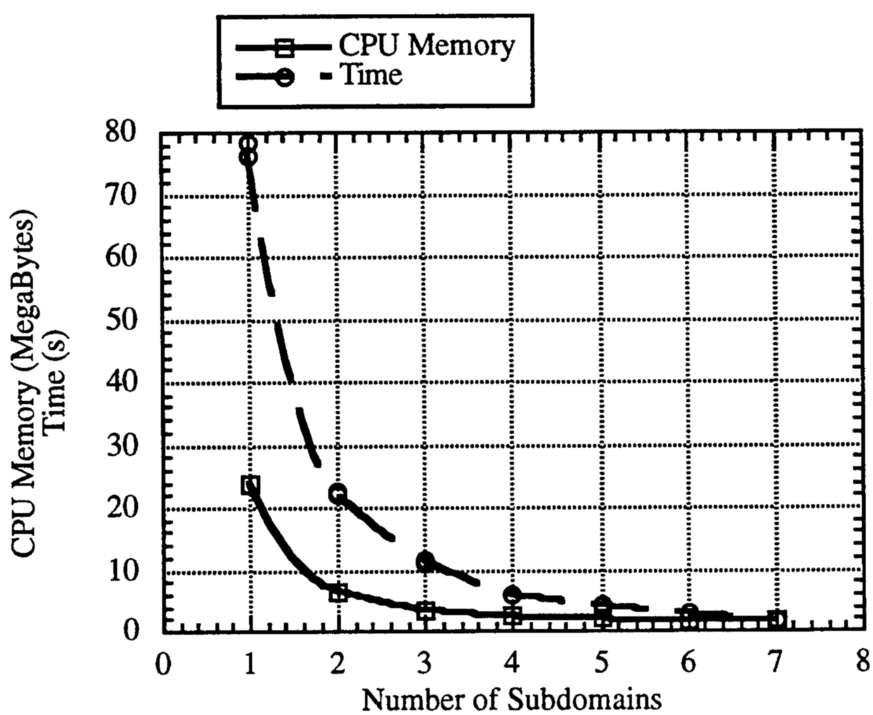


Figure 11-41. Influence of Subdomaining on the Memory Requirement and CPU Time for Thin-Plate Spline Method (Test 1p)

Table 11.16 Statistical Summary of Test Set 1 using Thin-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	1.95E-14	3.91E-13	8	1	3.92E-15	7.83E-14	4.94E-15	1.01
test1b	1.38E-14	2.75E-13	9	1	2.75E-15	5.51E-14	4.66E-15	1.3625
test1c	1.44E-14	7.33E-13	9	1	2.06E-15	1.05E-13	3.37E-15	1.44625
test1d	1.51E-14	3.02E-13	48	1	3.32E-15	6.63E-14	5.60E-15	0.10625
test1e	1.02E-06	2.04E-05	27	10	2.99E-07	5.99E-06	5.55E-07	0.28875
test1f	2.84E-02	1.422	36	11	6.71E-03	0.3363	9.49E-03	0.46062
test1g	1.87E-14	3.73E-13	14	1	6.34E-15	1.27E-13	3.00E-04	0.60031
test1h	9.22E-07	1.85E-05	12	11	2.09E-07	4.17E-06	9.51E-06	0.64516
test1i	2.84E-02	1.422	15	10	6.71E-03	0.3363	1.26E-02	0.73109
test1j	1.95E-14	3.91E-13	31	1	5.69E-15	1.14E-13	3.97E-04	0.76641
test1k	9.94E-07	1.99E-05	22	16	2.52E-07	5.04E-06	1.26E-05	0.87078
test1l	2.84E-02	1.431	20	15	6.61E-03	0.333	1.10E-02	0.86367
test1m	1.28E-13	2.56E-13	1	10	2.67E-14	5.33E-14	1.11E-03	2.57422
test1n	3.11E-15	3.05E-13	2	10	7.06E-16	6.92E-14	1.11E-04	2.59008
test1o	3.82E-17	1.91E-13	8	1	9.07E-18	4.54E-14	1.12E-05	2.59594
test1p	0.1316	6.597	1	2	2.34E-02	1.172	4.08E-02	9.78234
test1q	0.3873	19.37	85	4	0.1211	6.054	0.2026	12.4068
test1s	1.32E-03	6.597	70	2	2.34E-04	1.172	5.43E-03	9.54176
test1t	3.87E-03	19.37	85	17	1.21E-03	6.054	2.03E-03	12.14828
test1a1	2.22E-16	4.44E-13	1	10	3.53E-17	7.06E-14	2.04E-04	2.96539
test1b1	1.58E-16	3.16E-13	9	1	2.99E-17	5.98E-14	2.05E-05	2.96848
test1c1	2.21E-16	4.48E-13	10	10	5.50E-17	1.12E-13	2.06E-06	2.97156
test1d1	2.22E-16	4.44E-13	1	20	3.66E-17	7.33E-14	6.52E-08	1.32582
test1e1	6.02E-09	1.21E-05	42	11	1.27E-09	2.54E-06	3.02E-09	1.33215
test1f1	2.72E-04	1.359	12	2	4.53E-05	0.2268	8.60E-05	1.34863
test1g1	2.22E-16	4.44E-13	1	16	2.58E-17	5.15E-14	2.72E-06	1.36256
test1h1	4.66E-09	9.32E-06	12	11	1.27E-09	2.53E-06	8.61E-08	1.36069
test1i1	2.84E-04	1.422	15	10	6.71E-05	0.3363	1.26E-04	1.37889
test1j1	2.22E-16	4.44E-13	1	18	5.32E-17	1.06E-13	3.97E-06	1.38719
test1k1	8.07E-09	1.61E-05	23	15	1.49E-09	2.97E-06	1.26E-07	1.40536
test1l1	2.84E-04	1.431	20	15	6.61E-05	0.333	1.10E-04	1.3936
test1a2	1.95E-14	3.91E-13	8	1	3.92E-15	7.83E-14	4.94E-15	1.045
test1b2	1.95E-14	3.91E-13	9	1	3.18E-15	6.36E-14	5.25E-15	1.3675
test1c2	1.07E-14	5.41E-13	2	1	2.64E-15	1.34E-13	4.45E-15	1.45
test1d2	1.51E-14	3.02E-13	48	1	3.32E-15	6.63E-14	5.60E-15	0.0275
test1e2	9.44E-07	1.89E-05	15	17	3.86E-07	7.72E-06	6.41E-07	0.26125
test1f2	2.64E-02	1.322	22	16	5.70E-03	0.2849	8.88E-03	0.465
test1g2	1.87E-14	3.73E-13	14	1	6.34E-15	1.27E-13	2.81E-04	0.13562
test1h2	9.48E-07	1.90E-05	38	4	2.51E-07	5.02E-06	4.40E-07	0.405
test1i2	2.64E-02	1.322	29	5	5.70E-03	0.2849	1.07E-02	0.13375
test1j2	1.95E-14	3.91E-13	31	1	5.69E-15	1.14E-13	6.52E-15	0.44
test1k2	6.47E-07	1.29E-05	13	11	2.64E-07	5.29E-06	4.88E-07	0.13812
test1l2	2.57E-02	1.287	12	9	5.73E-03	0.2873	9.70E-03	0.55
test1m2	1.28E-13	2.56E-13	1	10	2.67E-14	5.33E-14	3.70E-14	1.045
test1n2	3.11E-15	3.05E-13	9	1	6.33E-16	6.20E-14	3.88E-15	1.3425
test1o2	2.43E-17	1.21E-13	10	10	6.61E-18	3.31E-14	3.90E-16	1.4625
test1p2	0.396	19.87	70	2	0.1058	5.31	0.1793	10.81625
test1a12	2.22E-16	4.44E-13	1	10	3.53E-17	7.06E-14	4.93E-17	1.025
test1b12	1.85E-16	3.70E-13	1	10	4.34E-17	8.67E-14	6.19E-17	1.375
test1c12	3.73E-16	7.58E-13	1	10	8.98E-17	1.82E-13	1.55E-16	1.48125
test1d12	2.22E-16	4.44E-13	1	20	3.66E-17	7.33E-14	5.66E-17	0.18125
test1e12	7.13E-09	1.43E-05	18	17	1.36E-09	2.72E-06	2.38E-09	0.12687
test1f12	2.84E-04	1.424	48	5	4.59E-05	0.2303	8.60E-05	0.31063
test1g12	2.22E-16	4.44E-13	1	16	2.58E-17	5.15E-14	2.72E-06	0.46719
test1h12	3.79E-09	7.57E-06	28	16	1.40E-09	2.80E-06	8.62E-08	0.48656
test1i12	2.64E-04	1.322	29	5	5.70E-05	0.2849	1.07E-04	0.60578
test1j12	2.22E-16	4.44E-13	1	18	5.32E-17	1.06E-13	3.37E-06	0.68344
test1k12	5.02E-09	1.00E-05	13	10	1.51E-09	3.01E-06	1.07E-07	0.72055
test1l12	2.57E-04	1.287	12	9	5.73E-05	0.2873	9.70E-05	0.76445

Table 11.16 Statistical Summary of Test Set 1 using Thin-Plate Splines (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1aa	5.00E-07	9.69E-06	6	7	1.80E-07	3.49E-06	1.81E-07	0.995
test1bb	8.84E-07	1.77E-05	7	4	2.03E-07	4.07E-06	2.05E-07	1.3475
test1cc	6.31E-07	3.08E-05	4	2	1.35E-07	6.60E-06	1.38E-07	1.4525
test1dd	7.21E-07	1.39E-05	19	9	2.24E-07	4.31E-06	2.24E-07	0.5275
test1ee	1.21E-06	2.42E-05	32	19	3.39E-07	6.78E-06	3.38E-07	0.15125
test1ff	2.95E-02	1.408	36	11	7.31E-03	0.349	9.57E-03	0.03969
test1gg	7.21E-07	1.39E-05	32	12	2.24E-07	4.31E-06	3.03E-04	0.14969
test1hh	1.15E-06	2.29E-05	12	6	2.28E-07	4.56E-06	9.58E-06	0.21125
test1ii	2.95E-02	1.423	15	10	7.31E-03	0.3526	9.57E-03	0.28328
test1jj	6.52E-07	1.26E-05	17	11	2.34E-07	4.54E-06	3.03E-04	0.34469
test1kk	1.18E-06	2.36E-05	29	18	2.68E-07	5.37E-06	9.58E-06	0.36289
test1ll	2.96E-02	1.426	20	15	7.15E-03	0.3443	9.57E-03	0.41906
test1mm	4.70E-06	9.37E-06	5	5	1.47E-06	2.94E-06	9.61E-04	2.5625
test1nn	8.29E-07	7.06E-05	5	9	2.35E-07	2.00E-05	9.66E-05	2.56875
test1oo	6.60E-08	3.80E-05	4	5	8.05E-09	4.64E-06	9.71E-06	2.62531
test1pp	0.1256	5.831	70	2	2.44E-02	1.135	6.99E-02	11.35305
test1qq	0.4481	20.83	46	4	0.1232	5.727	0.2303	13.21781
test1ss	1.26E-03	0.6632	70	2	2.44E-04	0.129	6.19E-03	11.0386
test1tt	4.48E-03	2.308	46	4	1.23E-03	0.6347	2.31E-03	13.11993
test1bb2	1.38E-14	2.75E-13	9	1	2.75E-15	5.51E-14	4.46E-03	2.99211
test1cc2	1.44E-14	7.33E-13	9	1	2.06E-15	1.05E-13	4.49E-04	2.99504
test1dd2	1.51E-14	3.02E-13	48	1	3.32E-15	6.63E-14	1.42E-05	1.27926
test1ee2	1.02E-06	2.04E-05	27	10	2.99E-07	5.99E-06	7.14E-07	1.26586
test1ff2	2.84E-02	1.422	36	11	6.71E-03	0.3363	9.49E-03	1.29631
test1gg2	1.87E-14	3.73E-13	14	1	6.34E-15	1.27E-13	3.00E-04	1.29471
test1hh2	9.22E-07	1.85E-05	12	11	2.09E-07	4.17E-06	9.51E-06	1.31291
test1ii2	2.84E-02	1.422	15	10	6.71E-03	0.3363	1.26E-02	1.32104
test1jj2	1.95E-14	3.91E-13	31	1	5.69E-15	1.14E-13	3.97E-04	1.32941
test1kk2	9.94E-07	1.99E-05	22	16	2.52E-07	5.04E-06	1.26E-05	1.30771
test1ll2	2.84E-02	1.431	20	15	6.61E-03	0.333	1.10E-02	1.3161
test1mm2	1.28E-13	2.56E-13	1	10	2.67E-14	5.33E-14	1.11E-03	3.08376
test1nn2	3.11E-15	3.05E-13	2	10	7.06E-16	6.92E-14	1.11E-04	3.09521
test1oo2	3.82E-17	1.91E-13	8	1	9.07E-18	4.54E-14	1.12E-05	3.07672
test1pp2	0.1316	6.597	1	2	2.34E-02	1.172	4.08E-02	9.38034
test1qq2	0.3873	19.37	85	4	0.1211	6.054	0.2026	11.6387
test1ss2	1.32E-03	6.597	70	2	2.34E-04	1.172	5.43E-03	9.31105
test1tt2	3.87E-03	19.37	85	17	1.21E-03	6.054	2.03E-03	11.52925
test1A	4.35E-14	4.35E-13	9	1	6.42E-15	6.42E-14	1.02E-14	1.005
test1B	9.92E-15	1.42E-13	10	10	2.72E-15	3.90E-14	3.80E-15	1.36
test1C	1.49E-06	1.49E-05	17	5	3.91E-07	3.91E-06	6.61E-07	0.25125
test1D	1.47E-06	1.47E-05	29	19	3.56E-07	3.56E-06	6.19E-07	0.16
test1E	2.84E-02	0.4069	20	15	6.61E-03	9.47E-02	1.10E-02	0.32687
test1F	2.57E-02	0.367	39	9	5.73E-03	8.19E-02	9.71E-03	0.46
test1G	1.08E-06	5.27E-05	11	10	3.11E-07	1.53E-05	3.07E-04	0.55688
test1H	2.57E-02	0.8517	12	9	5.73E-03	0.1901	9.70E-03	0.61422
test1I	0.1532	3.836	4	20	2.88E-02	0.7206	4.81E-02	9.96156
test1J	0.223	11.04	26	2	1.04E-02	0.5137	3.92E-02	266.25922
test1A1	8.33E-07	8.33E-06	6	9	2.37E-07	2.37E-06	2.38E-07	0.975
test1B1	8.29E-07	1.19E-05	3	8	1.75E-07	2.51E-06	1.77E-07	1.3575
test1C1	1.71E-06	1.71E-05	11	19	4.17E-07	4.17E-06	4.17E-07	0.425
test1D1	1.92E-06	1.92E-05	15	9	3.79E-07	3.79E-06	3.79E-07	0.06563
test1E1	2.96E-02	0.4237	20	15	7.15E-03	0.1023	7.15E-03	0.19281
test1F1	2.70E-02	0.386	39	12	6.13E-03	8.76E-02	6.14E-03	0.29844
test1G1	1.20E-06	5.49E-05	26	3	3.10E-07	1.42E-05	1.94E-04	0.28219
test1H1	2.70E-02	0.8685	12	9	6.13E-03	0.1971	6.13E-03	0.34219
test1I1	0.1528	3.748	35	14	2.96E-02	0.7271	2.97E-02	11.37156
test1J1	0.2242	10.29	26	2	1.05E-02	0.4817	1.05E-02	272.85382

Table 11.16 Statistical Summary of Test Set 1 using Thin-Plate Splines (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1C2	5.16E-06	5.06E-05	3	7	6.30E-07	6.18E-06	3.41E-06	1.5097
test1D2	1.54E-06	1.54E-05	3	7	3.93E-07	3.93E-06	4.08E-07	1.48383
test1E2	2.84E-02	0.3984	20	15	6.61E-03	9.27E-02	6.62E-03	1.5181
test1F2	2.57E-02	0.3595	39	9	5.73E-03	8.02E-02	5.74E-03	1.5123
test1G2	1.03E-06	4.24E-05	35	13	3.12E-07	1.29E-05	1.82E-04	1.53653
test1H2	2.57E-02	0.7651	12	9	5.73E-03	0.1707	5.73E-03	1.5607
test1I2	0.1532	3.538	4	20	2.88E-02	0.6646	2.88E-02	9.19479
test1J2	0.223	9.246	45	99	1.04E-02	0.4302	1.22E-02	260.87589

Table 11.17 Statistical Summary of Test Set 2 using Thin-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test2A	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	1.34
test2B	4.00E-07	2.00E-05	1	5	2.00E-07	1.00E-05	3.18E-07	1.61
test2C	4.00E-07	2.00E-05	5	9	2.00E-07	1.00E-05	3.19E-07	1.715
test2D	8.00E-07	1.33E-05	3	5	2.46E-07	4.09E-06	4.37E-07	1.785
test2E	0.4423	22.46	8	7	0.1127	5.722	0.2013	1.83125
test2F	0.00E+00	0.00E+00	8	7	0.00E+00	0.00E+00	4.03E-03	0.71625
test2G	5.24E-07	2.62E-05	44	3	2.64E-07	1.32E-05	8.05E-05	1.23125
test2H	5.24E-07	2.62E-05	3	27	2.64E-07	1.32E-05	1.66E-06	1.44531
test2I	9.05E-07	1.51E-05	46	19	2.52E-07	4.20E-06	4.21E-07	1.54297
test2L	1.23E-09	1.23E-05	17	21	2.44E-10	2.44E-06	8.44E-09	1.63797
test2M	1.36E-09	1.36E-05	17	32	2.33E-10	2.33E-06	4.15E-10	1.72609
test2N	4.47E-04	22.52	32	29	7.97E-05	4.015	1.55E-04	1.77344
test2A1	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	1.295
test2B1	5.49E-07	2.75E-05	7	8	1.93E-07	9.64E-06	1.95E-07	1.62
test2C1	5.49E-07	2.75E-05	8	7	1.93E-07	9.64E-06	1.95E-07	1.7275
test2D1	9.75E-07	1.63E-05	3	3	2.32E-07	3.87E-06	2.34E-07	1.81
test2E1	0.4423	22.01	8	7	0.1127	5.61	0.1133	1.8275
test2F1	6.06E-07	2.78E-05	10	7	2.36E-07	1.09E-05	2.27E-03	0.76625
test2G1	6.06E-07	3.03E-05	23	16	2.39E-07	1.20E-05	4.53E-05	1.26656
test2H1	6.04E-07	3.02E-05	16	23	2.39E-07	1.20E-05	9.38E-07	1.41875
test2I1	1.02E-06	1.71E-05	48	18	2.52E-07	4.19E-06	2.52E-07	1.54141
test2L1	9.73E-08	5.30E-05	25	28	1.13E-09	6.14E-07	5.17E-09	1.62328
test2M1	9.73E-08	5.30E-05	28	26	1.15E-09	6.28E-07	1.16E-09	1.74781
test2N1	4.47E-04	0.2503	32	29	7.97E-05	4.46E-02	9.19E-05	1.77485

Table 11.18 Statistical Summary of Test Set 3 using Thin-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A	1.21E-06	2.41E-05	4	5	3.09E-07	6.19E-06	5.01E-07	0.995
test3B	0.4117	11.81	1	8	5.42E-02	1.556	0.1368	1.3475
test3C	1.39E-06	2.78E-05	2	11	3.94E-07	7.88E-06	4.33E-03	0.3425
test3D	1.07E-06	2.13E-05	9	4	2.85E-07	5.69E-06	1.37E-04	0.03125
test3E	0.8659	24.79	1	7	0.2354	6.74	0.5673	0.23906
test3F	0.4146	11.85	1	12	9.44E-02	2.7	0.2055	0.50719
test3G	1.34E-06	1.31E-04	3	15	3.70E-07	3.62E-05	6.50E-03	0.52875
test3H	0.4146	27.51	1	12	9.44E-02	6.264	0.2048	0.62594
test3I	2.294	114.9	1	6	0.2317	11.61	0.6706	11.07062
test3J	4.631	458.6	3	49	0.4947	49	1.215	392.47632
test3A1	1.00E-06	2.01E-05	4	3	2.96E-07	5.91E-06	2.97E-07	0.995
test3B1	0.4064	11.66	1	8	5.64E-02	1.619	0.302	1.36
test3C1	1.52E-06	3.04E-05	9	13	4.13E-07	8.26E-06	9.56E-03	0.4375
test3D1	1.14E-06	2.28E-05	7	2	2.77E-07	5.54E-06	3.02E-04	0.04625
test3E1	0.8592	24.6	1	20	0.2373	6.794	0.3373	0.11969
test3F1	0.4098	11.72	1	12	9.59E-02	2.742	0.3462	0.30844
test3G1	1.24E-06	1.06E-04	4	16	3.62E-07	3.09E-05	1.10E-02	0.36719
test3H1	0.4098	25.55	1	12	9.59E-02	5.979	9.59E-02	0.46531
test3I1	2.34	112.4	1	1	0.2773	13.32	2.618	12.43
test3J1	4.417	374.8	3	84	0.5241	44.47	2.735	402.02594
test3A2	1.07E-06	2.13E-05	4	6	3.13E-07	6.25E-06	3.14E-07	1
test3B2	0.4117	11.81	1	8	5.44E-02	1.562	0.2918	1.375
test3C2	1.56E-06	3.11E-05	9	13	4.24E-07	8.48E-06	9.23E-03	0.27
test3D2	1.18E-06	2.37E-05	7	2	2.86E-07	5.72E-06	2.92E-04	0.1225
test3E2	0.8659	24.79	1	7	0.2363	6.764	0.3374	0.31094
test3F2	0.4146	11.85	1	12	9.44E-02	2.7	0.3363	0.45344
test3G2	1.31E-06	1.12E-04	8	10	3.74E-07	3.19E-05	1.06E-02	0.47313
test3H2	0.4146	25.85	1	12	9.44E-02	5.887	9.45E-02	0.60906
test3I2	2.364	113.5	1	1	0.2334	11.21	2.598	11.16953
test3J2	4.631	392.9	3	49	0.4952	42.02	2.859	393.88522
test3A3	4.44E-16	2.22E-14	2	3	1.40E-16	6.99E-15	3.37E-16	1.35
test3B3	4.00E-07	2.00E-05	1	6	2.00E-07	1.00E-05	3.18E-07	1.62
test3C3	4.00E-07	2.00E-05	6	1	2.00E-07	1.00E-05	3.20E-07	1.725
test3D3	8.01E-07	1.34E-05	3	5	2.47E-07	4.12E-06	4.38E-07	1.805
test3E3	0.7094	36.02	10	10	0.1419	7.206	0.2426	1.8425
test3F3	4.44E-16	2.22E-14	21	17	1.04E-16	5.21E-15	4.85E-03	0.785
test3G3	5.24E-07	2.62E-05	18	3	2.64E-07	1.32E-05	9.71E-05	1.23906
test3H3	5.23E-07	2.62E-05	3	50	2.64E-07	1.32E-05	1.98E-06	1.45
test3I3	9.02E-07	1.50E-05	46	19	2.51E-07	4.18E-06	4.16E-07	1.56984
test3J3	0.7094	35.72	50	50	0.1848	9.303	0.2717	1.655
test3K3	1.04E-17	1.04E-13	23	26	4.34E-18	4.34E-14	5.44E-03	1.765
test3L3	1.54E-09	1.54E-05	3	6	3.45E-10	3.45E-06	1.09E-04	1.79125
test3M3	1.53E-09	1.53E-05	3	43	3.03E-10	3.03E-06	2.18E-06	1.83852
test3N3	7.09E-04	35.72	50	50	1.85E-04	9.303	2.72E-04	1.875
test3A4	5.44E-07	2.52E-05	2	2	1.83E-07	8.49E-06	1.84E-07	1.34
test3B4	5.49E-07	2.75E-05	7	8	1.93E-07	9.64E-06	1.95E-07	1.6
test3C4	5.49E-07	2.75E-05	8	7	1.93E-07	9.65E-06	1.95E-07	1.695
test3D4	9.75E-07	1.63E-05	3	3	2.35E-07	3.92E-06	2.32E-07	1.785
test3E4	0.7094	35.31	10	10	0.1458	7.259	0.5398	1.8425
test3F4	6.06E-07	2.78E-05	10	7	2.36E-07	1.09E-05	1.08E-02	0.81
test3G4	6.06E-07	3.03E-05	23	16	2.39E-07	1.20E-05	2.16E-04	1.23156
test3H4	6.04E-07	3.02E-05	16	23	2.39E-07	1.20E-05	4.33E-06	1.43531
test3I4	1.02E-06	1.70E-05	48	18	2.50E-07	4.17E-06	2.60E-07	1.52516
test3J4	0.7094	34.71	50	50	0.185	9.049	0.3957	1.60703
test3K4	4.78E-08	2.54E-05	28	29	6.08E-10	3.23E-07	7.92E-03	1.72891
test3L4	9.74E-08	5.30E-05	25	28	1.21E-09	6.60E-07	1.58E-04	1.77594
test3M4	9.73E-08	5.30E-05	28	26	1.22E-09	6.64E-07	3.17E-06	1.84289
test3N4	7.09E-04	0.3968	50	50	1.85E-04	0.1035	3.57E-04	0.6075

Table 11.19 Statistical Summary of Test Set 4 using Thin-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test4A2	1.14E-06	2.28E-05	11	22	4.28E-07	8.56E-06	6.54E-07	4.56
test4A3	1.43E-06	2.86E-05	28	31	4.38E-07	8.76E-06	7.06E-07	25.14
test4B2	7.12E-02	2.048	38	29	2.48E-02	0.7137	3.99E-02	34.11
test4B3	7.17E-02	2.061	76	58	2.51E-02	0.7222	4.01E-02	54.62
test4C2	7.86E-07	1.57E-05	129	65	4.71E-07	9.42E-06	5.90E-07	47.15
test4C3	1.04E-06	2.08E-05	317	46	5.16E-07	1.03E-05	5.81E-07	246.83
test4D2	1.32E-06	2.63E-05	121	32	4.67E-07	9.34E-06	6.32E-07	46.92
test4D3	1.42E-06	2.83E-05	54	70	4.70E-07	9.39E-06	6.32E-07	246.53
test4E2	7.18E-02	2.063	188	59	1.71E-02	0.492	3.53E-02	46.9
test4E3	7.18E-02	2.062	376	117	1.72E-02	0.4931	3.54E-02	247.71001
test4F2	6.96E-02	1.999	87	46	1.45E-02	0.4177	2.68E-02	344.76999
test4F3	7.15E-02	2.058	173	69	1.47E-02	0.4233	2.70E-02	547.67999
test4G2	1.05E-06	1.03E-04	160	48	4.72E-07	4.63E-05	7.01E-07	47.02
test4G3	1.04E-06	1.02E-04	380	54	4.85E-07	4.76E-05	7.14E-07	246.34
test4H2	6.96E-02	4.665	87	46	1.45E-02	0.9747	2.68E-02	339.17001
test4H3	7.15E-02	4.812	173	69	1.47E-02	0.9903	2.70E-02	539.47998
test4A12	5	100	1	1	2.5	50	5.119	4.92
test4A13	5	100	1	1	2.5	50	5.111	27.28
test4B12	2.894	83.25	1	40	1.176	33.83	1.484	36.67
test4B13	2.893	83.17	1	80	1.172	33.69	1.226	59.03
test4C12	5	100	1	80	4.535	90.71	9.093	114.74
test4C13	5	100	1	160	4.537	90.74	9.095	330.67001
test4D12	5	100	1	1	3.285	65.71	6.685	427.48999
test4D13	5	100	1	1	3.287	65.74	6.686	643.87
test4E12	2.893	83.13	200	1	1.278	36.72	2.565	740.91998
test4E13	2.893	83.08	400	1	1.278	36.71	2.547	270.56
test4F12	2.893	83.1	1	80	1.947	55.92	2.026	51.34
test4F13	2.893	83.22	1	160	1.948	56.02	1.988	267.35999
test4G12	1.02	84.64	200	80	1.013	84.1	2.027	364.22
test4G13	1.02	84.32	400	160	1.013	83.77	2.027	579.63
test4H12	0.6545	41.47	192	38	0.4115	26.07	0.9345	676.58002
test4H13	0.6577	41.82	385	76	0.4107	26.11	0.9301	894.71002

Table 11.20 Statistical Summary of Test Set 5 using Thin-Plate Splines without Scaling the Data

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test5a	6.66E-16	6.66E-14	7	1	0.00E+00	0.00E+00	2.00E-16	1.4
test5b	2.78E-16	2.78E-14	9	1	0.00E+00	0.00E+00	1.59E-16	1.515
test5c	8.33E-16	8.33E-14	5	1	1.39E-16	1.39E-14	7.85E-16	1.605
test5d	2.55E-15	2.55E-13	70	1	1.74E-17	1.74E-15	8.71E-16	1.29
test5e	2.67E-15	2.67E-13	59	1	1.52E-17	1.52E-15	7.64E-16	1.455
test5f	1.03E-14	1.03E-12	73	1	3.47E-18	3.47E-16	3.03E-15	1.56625
test5g	8.33E-05	8.33E-03	3	1	4.44E-16	4.44E-14	1.35E-05	0.8425
test5h	2.68E-04	2.68E-02	8	1	2.22E-16	2.22E-14	1.02E-04	1.06938
test5i	7.32E-04	7.32E-02	8	1	2.22E-16	2.22E-14	3.24E-04	1.25281
test5j	2.00E-15	2.00E-12	1	1	0.00E+00	0.00E+00	1.08E-04	2.53719
test5k	4.86E-17	4.86E-14	2	1	0.00E+00	0.00E+00	3.60E-05	2.56
test5l	5.55E-17	5.55E-14	5	1	0.00E+00	0.00E+00	1.20E-05	2.55156
test5m	4.79E-16	4.79E-13	59	1	8.67E-19	8.67E-16	1.21E-06	2.1075
test5n	5.41E-16	5.41E-13	54	1	6.07E-18	6.07E-15	1.21E-07	2.11281
test5o	1.42E-15	1.42E-12	65	1	0.00E+00	0.00E+00	1.22E-08	2.17906
test5p	8.33E-06	8.33E-03	3	1	0.00E+00	0.00E+00	1.35E-06	1.37156
test5k	4.86E-17	4.86E-14	2	1	0.00E+00	0.00E+00	4.49E-07	2.7075
test5r	7.32E-05	7.32E-02	8	1	1.39E-17	1.39E-14	3.24E-05	1.47156
test5a1	8.88E-16	2.96E-14	10	1	0.00E+00	0.00E+00	1.08E-05	2.74859
test5b1	7.11E-15	2.37E-13	10	1	4.44E-16	1.48E-14	3.60E-06	2.75047
test5c1	7.11E-15	2.37E-13	1	1	6.66E-16	2.22E-14	1.20E-06	2.75203
test5d1	3.38E-14	1.13E-12	100	1	2.08E-17	6.94E-16	1.21E-07	2.31594
test5e1	4.04E-14	1.35E-12	100	1	8.33E-17	2.78E-15	1.21E-08	2.32937
test5f1	4.04E-14	1.35E-12	1	1	0.00E+00	0.00E+00	1.22E-09	2.32297
test5g1	2.49E-03	8.31E-02	498	1	8.88E-15	2.96E-13	1.87E-04	1.56
test5h1	8.40E-03	0.2799	498	1	3.86E-14	1.29E-12	6.61E-04	1.54922
test5i1	4.92E-03	0.1639	498	1	3.20E-14	1.07E-12	7.64E-04	1.60922
test5j1	3.20E-14	1.07E-11	1	1	0.00E+00	0.00E+00	2.55E-04	2.91313
test5k1	1.11E-16	3.70E-14	10	1	0.00E+00	0.00E+00	8.49E-05	2.92453
test5l1	2.64E-16	8.79E-14	7	1	5.55E-17	1.85E-14	2.83E-05	2.91562
test5m1	8.33E-16	2.78E-13	56	1	8.63E-18	2.88E-15	2.84E-06	2.46953
test5n1	5.69E-16	1.90E-13	72	1	1.83E-17	6.11E-15	2.86E-07	2.49312
test5o1	1.76E-15	5.88E-13	72	1	0.00E+00	0.00E+00	2.87E-08	2.54641
test5p1	3.15E-05	1.05E-02	498	1	0.00E+00	0.00E+00	4.49E-06	1.70219
test5k1	1.11E-16	3.70E-14	10	1	0.00E+00	0.00E+00	1.50E-06	2.98265
test5r1	1.44E-04	4.81E-02	8	1	1.11E-16	3.70E-14	6.60E-05	1.73625

Table 11.21 Statistical Summary of Test Set 5 using Thin-Plate Splines and Scaling the Data

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test5a	6.66E-16	6.66E-14	7	1	0.00E+00	0.00E+00	2.00E-16	1.37
test5b	2.78E-16	2.78E-14	9	1	0.00E+00	0.00E+00	1.59E-16	1.495
test5c	8.33E-16	8.33E-14	5	1	1.39E-16	1.39E-14	7.85E-16	1.565
test5d	3.44E-15	3.44E-13	56	1	0.00E+00	0.00E+00	4.13E-15	1.22
test5e	1.55E-15	1.55E-13	57	1	0.00E+00	0.00E+00	5.86E-16	1.43
test5f	1.13E-14	1.13E-12	58	1	0.00E+00	0.00E+00	1.34E-14	1.57
test5g	8.33E-05	8.33E-03	3	1	0.00E+00	0.00E+00	1.35E-05	0.8275
test5h	2.68E-04	2.68E-02	8	1	4.44E-16	4.44E-14	1.02E-04	1.08187
test5i	7.32E-04	7.32E-02	8	1	6.66E-16	6.66E-14	3.24E-04	1.20406
test5j	1.11E-15	1.11E-12	1	1	0.00E+00	0.00E+00	1.08E-04	2.56094
test5k	4.86E-17	4.86E-14	2	1	0.00E+00	0.00E+00	3.60E-05	2.53312
test5l	5.55E-17	5.55E-14	5	1	0.00E+00	0.00E+00	1.20E-05	2.57656
test5m	2.85E-16	2.85E-13	62	1	0.00E+00	0.00E+00	1.21E-06	2.11281
test5n	4.37E-16	4.37E-13	63	1	0.00E+00	0.00E+00	1.21E-07	2.13844
test5o	1.94E-16	1.94E-13	32	1	0.00E+00	0.00E+00	1.22E-08	2.16406
test5p	8.33E-06	8.33E-03	3	1	0.00E+00	0.00E+00	1.35E-06	1.37594
test5k	4.86E-17	4.86E-14	2	1	0.00E+00	0.00E+00	4.49E-07	2.71125
test5r	7.32E-05	7.32E-02	8	1	1.39E-17	1.39E-14	3.24E-05	1.44594
test5a1	1.78E-15	5.92E-14	10	1	0.00E+00	0.00E+00	1.08E-05	2.77156
test5b1	1.78E-15	5.92E-14	1	1	0.00E+00	0.00E+00	3.60E-06	2.74281
test5c1	1.78E-15	5.92E-14	1	1	0.00E+00	0.00E+00	1.20E-06	2.76422
test5d1	4.13E-14	1.38E-12	100	1	1.80E-16	6.01E-15	1.21E-07	2.2975
test5e1	4.13E-14	1.38E-12	1	1	2.22E-16	7.40E-15	1.21E-08	2.3
test5f1	4.04E-14	1.35E-12	100	1	0.00E+00	0.00E+00	1.22E-09	2.36359
test5g1	2.17E-03	7.23E-02	498	1	3.33E-14	1.11E-12	1.64E-04	1.54984
test5h1	7.27E-03	0.2422	498	1	3.33E-14	1.11E-12	5.85E-04	1.52922
test5i1	4.57E-03	0.1524	498	1	1.04E-13	3.46E-12	7.51E-04	1.62016
test5j1	1.04E-13	3.46E-11	1	1	3.47E-18	1.16E-15	2.50E-04	2.94391
test5k1	2.78E-16	9.25E-14	10	1	2.78E-17	9.25E-15	8.34E-05	2.91469
test5l1	2.78E-16	9.25E-14	1	1	0.00E+00	0.00E+00	2.78E-05	2.89609
test5m1	2.55E-15	8.51E-13	100	1	3.25E-19	1.08E-16	2.79E-06	2.48985
test5n1	2.55E-15	8.51E-13	1	1	0.00E+00	0.00E+00	2.81E-07	2.53281
test5o1	2.44E-15	8.14E-13	1	1	0.00E+00	0.00E+00	2.82E-08	2.51516
test5p1	2.17E-04	7.23E-02	498	1	1.94E-15	6.48E-13	1.64E-05	1.72203
test5k1	2.00E-15	6.66E-13	1	1	2.78E-17	9.25E-15	5.46E-06	2.97242
test5r1	4.57E-04	0.1524	498	1	2.78E-16	9.25E-14	7.50E-05	1.74609

11.5 Finite-Plate Spline Method

The Finite-Plate Spline (FPS) method is a finite-element based approach where a virtual mesh is created in the interface between the input and output grids. The method tested here is based on the formulation presented by Appa [14], and the mathematical details are presented in Chapter 8.

The implementation of FPS was based the two-dimensional implementation by Dr. Kari Appa. Even though the method can be extended to three dimensions by using shell elements or plate elements in space, the version available for these tests is restricted to two dimensions. Therefore, all the results presented herein are restricted to plate cases. FPS is implemented so that the virtual mesh is generated automatically, requiring only the maximum number of nodes in the mesh. The mesh is defined by the aspect ratio of the geometric domain, and does not consider the gradient of the function along each direction. For a square grid, the same number of elements would be assigned to both directions. In order to better use the limited discretization available, the virtual mesh was set up such that it has 19 elements in one direction and 2 in the other. The finer discretization was used along the direction of the most rapidly varying function.

<u>Overall Accuracy</u> – Statistical summaries of the FPS results are presented in Tables 11.22 – 11.25. The overall accuracy of FPS is very good for the test cases examined. The error does not vary across the range of test case functions, and is usually less than 1% of the input function magnitude. The largest errors occurred in the sinusoidal cases with high number of cycles. Figures 11-42 and 11-43 plot the errors for each category of function test cases. It is directly observed from these plots that the method has excellent performance for constant, linear and for one-cycle sinusoidal functions. There is, however, a tendency towards higher errors for the high-frequency sinusoidal functions. For the majority of the runs, there are two orders of magnitude difference between the maximum and the average errors, indicating that the method is sensitive to particular locations on the grid. The interpolation errors tend to be the highest along the edges of the surface, as shown in Figure 11-44. The primary disadvantage of this method is its high CPU time and memory requirements.

<u>Grid Spacing Sensitivity</u> – Figure 11-42 indicates that the sensitivity of the FPS method to grid spacing is invariant with the function to be interpolated. The "A" runs on the figure indicate that the function was interpolated to an identical grid, while "B" runs indicate interpolation to a clustered grid. The error increased 0.5 and 2 orders of magnitude between these two types of grids, with the maximum errors less than 10% of the function magnitude.

<u>Virtual Mesh Sensitivity</u> – As discussed previously, FPS requires the definition of a virtual mesh that will cover the interface between the structural and the aerodynamic grids. The code is designed so that the mesh is generated automatically. It was originally designed to define the mesh based on the aspect ratio of the geometric domain, without considering the gradient of the function along each direction. So, for the test cases that have unit square domains, the number of elements along each direction was the same. This in a sense is a misuse of the virtual surface. In order to see the influence of increasing the number of elements along one direction while keeping the total number of nodes constant, an alternative mesh was created. The number of elements in each direction was set up *a priori*, and it was based on 19 elements along the highest gradient direction, and only 2 elements along the other direction (the automatic one generates a 6×6 mesh, since the maximum number of elements was 60). Figure 11-43 indicates that the sensitivity of FPS to virtual meshing definition, which is also associated with the function to be interpolated. For all of the test cases plotted in these figures, the data are similar to those in Figure 11-42. The difference here is that the function variation changes directions (test cases1a2 – 12) while the discretization of the virtual mesh stays the same. The error increases mainly for the sinusoidal functions, as

expected, indicating that the method is sensitive to the way the virtual mesh is oriented. For future use of this method, special attention must be paid to the issue of automated mesh generation.

Directional Bias – Figure 11-42 includes the streamwise (test1a – 11) functions examined in test set 1. The spanwise (test1a2 – 12) functions give virtually identical results when the virtual mesh is aligned so that the finer mesh is along the variation of the function. This indicates little or no sensitivity to direction of the function when the proper discretization of the virtual mesh is applied. This is important since the orientation of the surface to be interfaced is not known, and the dominant function (the data which change the most) can be located in either direction on the surface.

Magnitude/Amplitude Sensitivity – FPS appears to have no relative sensitivity to different magnitudes, as shown in Figure 11-45. This trend is consistent for function type, plates, direction of the function and grid (test sets 1 and 2).

Sensitivity to Frequency (Higher Oscillations) – FPS is sensitive to the frequency of the function. In Tables 11.22 – 11.25, the errors are very low for constant and linearly varying functions. Figures 11-42 and 11-43 substantiate this trend. There is a large jump in error when a sinusoidally-varying function is introduced. The sinusoidal functions shown in Figures 11-42 and 11-43 are for one cycle over the surface. When the frequency of the sinusoidal function is increased, the error of the FPS interpolations increases quite significantly. Figure 11-46 shows the results for these higher-frequency oscillations. A variation of 3 orders of magnitude is seen when the function increases from one to three cycles, and another order of magnitude when it increases from three to five cycles. However, at five cycles, the relative error is only about 1% of the maximum amplitude (refer to Tables 11.22 and 11.23). The errors remain insignificant, as illustrated in Figure 11-47 for a three-cycle sinusoidal function. The method seems to be quite accurate, and the final impact of the error increasing is not too dramatic.

Extrapolation – The maximum errors for test sets 1 and 3 are plotted in Figure 11-48, illustrating that FPS preserves its performance for extrapolations. There is a maximum one order of magnitude difference between the interpolation (represented by "A") and extrapolation (represented by "B") results, as confirmed by Table 11.24.

Diminishing Variation – Due to CPU memory limitations, this study could not be performed with the workstations available (see Memory Requirements below).

Sensitivity to Grid Irregularities – For the irregular grid test set (test set 2), FPS performed very well for the constant and linearly-varying functions (<<0.01%). There is a large jump in the error when it comes to the sinusoidal functions (~20%) as seen in Table 11.23. The virtual mesh is very sparse, and it seems likely that this caused the error to increase even more. Therefore, it is important that this method be used with input and virtual grids that have enough points to adequately identify the function shape.

Sensitivity to Planar Curves (Beam-Like Cases) – Table 11.25 summarizes the statistical results obtained from FPS when interpolating data along a planar curve. The method performed well and the maximum error was approximately 1%. Since the method is based on a two-dimensional formulation, a two-dimensional virtual mesh was created that encompassed the one-dimensional domain. FPS has been shown previously to be sensitive to the virtual mesh definition. The discretization used in the results presented on Table 11.25 was based on a square mesh, with the same dimensions as the length of the one-dimensional domain.

Algorithm CPU Memory and Time Requirements – The average CPU time requirement is approximately 20–80 seconds (refer to the last column of Tables 11.22 through 11.25), with the exception of the test cases which had very large errors, indicative of ill-conditioned matrices for

some of the sinusoidal functions. These runs required on the average of 500 seconds, reaching convergence at the end. The code used to run these cases required between 120 to 180MB of memory.

The algorithm CPU memory requirement was determined by varying the different matrix sizes for the known and unknown function grids, keeping the maximum number of elements in the virtual mesh fixed to 60. The results of this variation are shown in Figure 11-49. The results were curve fit using a second degree polynomial and the following memory algorithm was derived :

$$\text{CPU SIZE (MBytes)} = 7.78 + 7.9509\times10^{-5}\,\text{KGS} + 4.5026\times10^{-11}\,\text{KGS}^2 + 1.1590\times10^{-4}\,\text{UKS} + 8.0\times10^{-11}\,\text{UKS}^2$$

where KGS is the total number of grid points in the starting or known function grid and UKS is the total number of grid points in the grid to be interpolated to. This function yields values within 1% of the actual CPU size during testing. As one can see, there is a large increase in the memory requirement when one goes beyond 200 points in the grid to be interpolated. This mesh size is still small for practical purpose applications and the CPU requirement is unacceptable for most current workstations. Besides the fact that this method is a finite-element based approach, and therefore is well-known for high memory requirements, the set of routines used does not seem to take full advantage of sparseness in certain matrices.

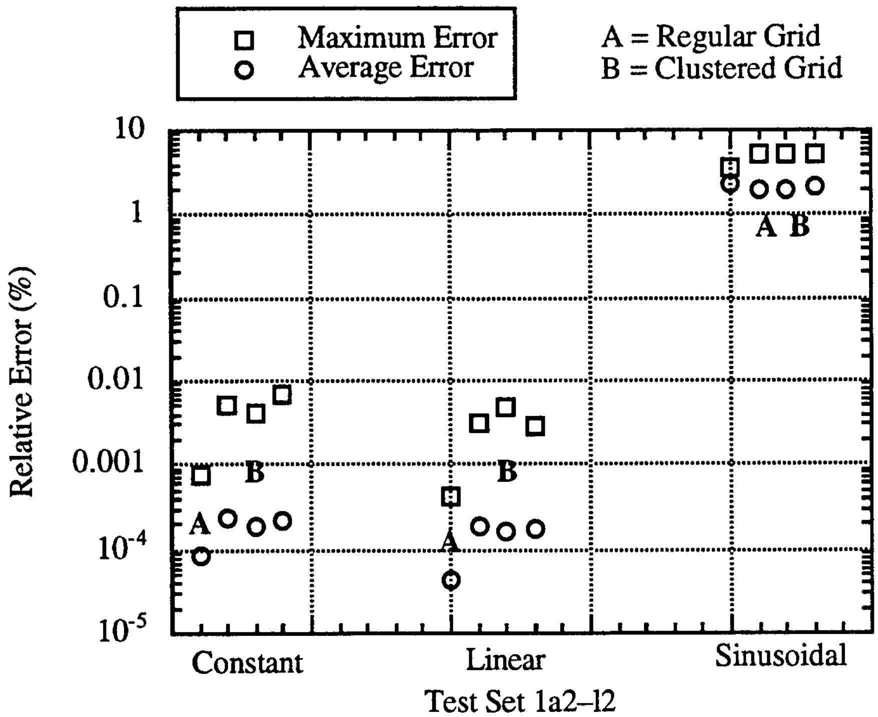


Figure 11-42. Variation of Error for Test Set One Based Upon Function Type for Plates (refined mesh along the varying function) Using Finite-Plate Splines

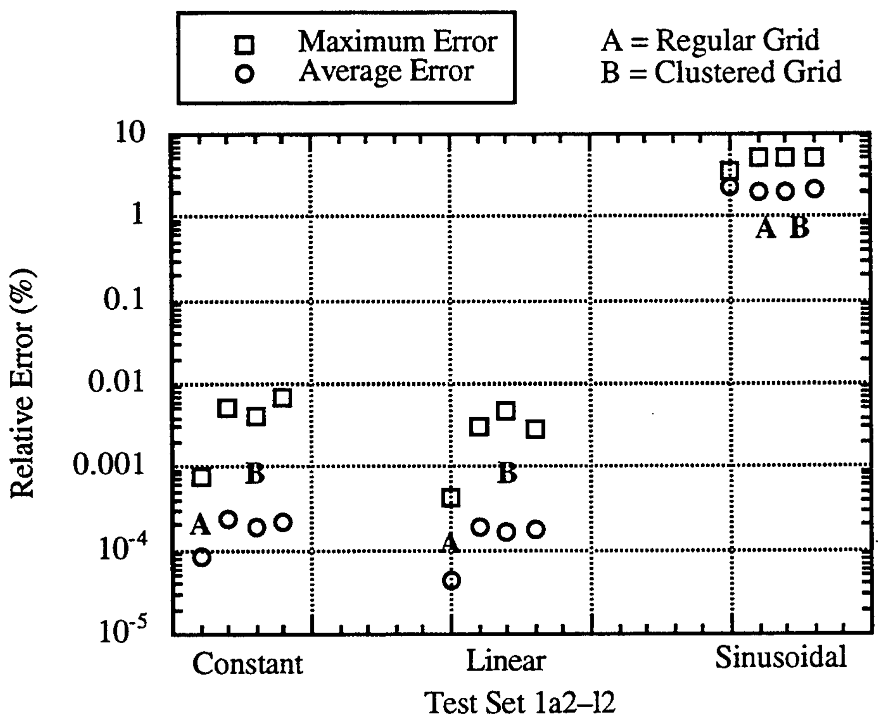
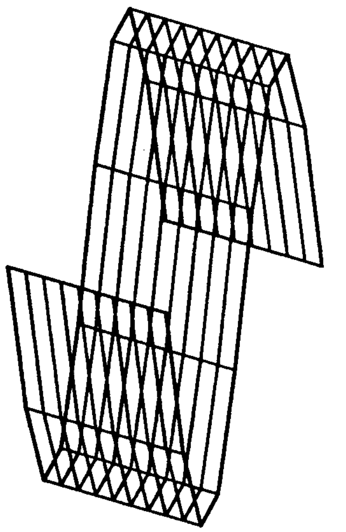
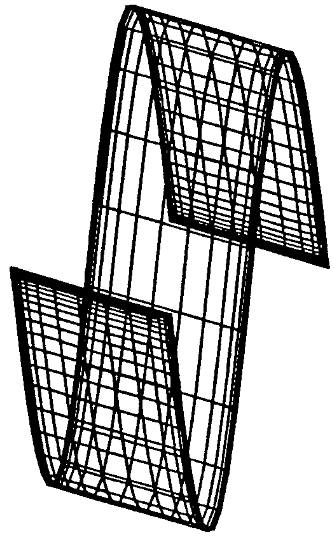


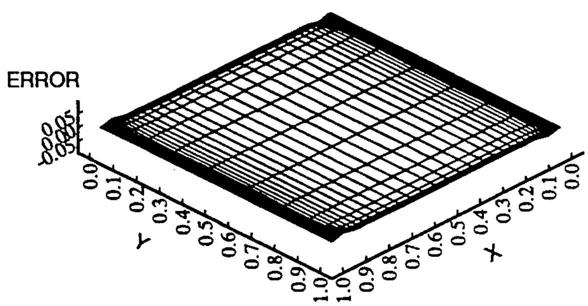
Figure 11-43. Variation of Error for Test Set One Based Upon Function Type for Plates (crude mesh along the varying function) Using Finite-Plate Splines



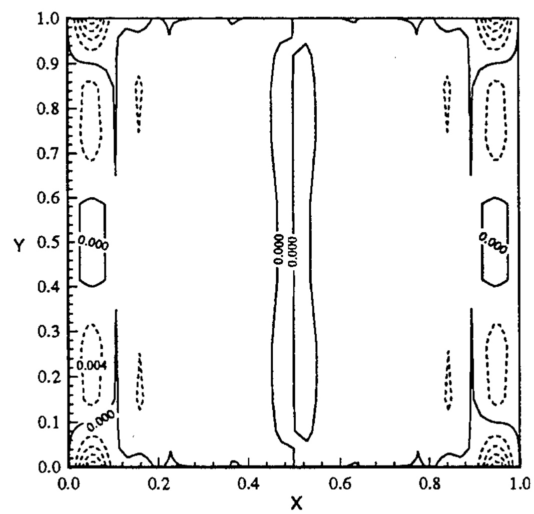
a) Original Function



b) Interpolated Function



c) Orthogonal View of the Error



d) Error Contours

Figure 11-44. Example of Oscillations Induced by the Finite-Plate Spline Method (Test 11) for a One-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2

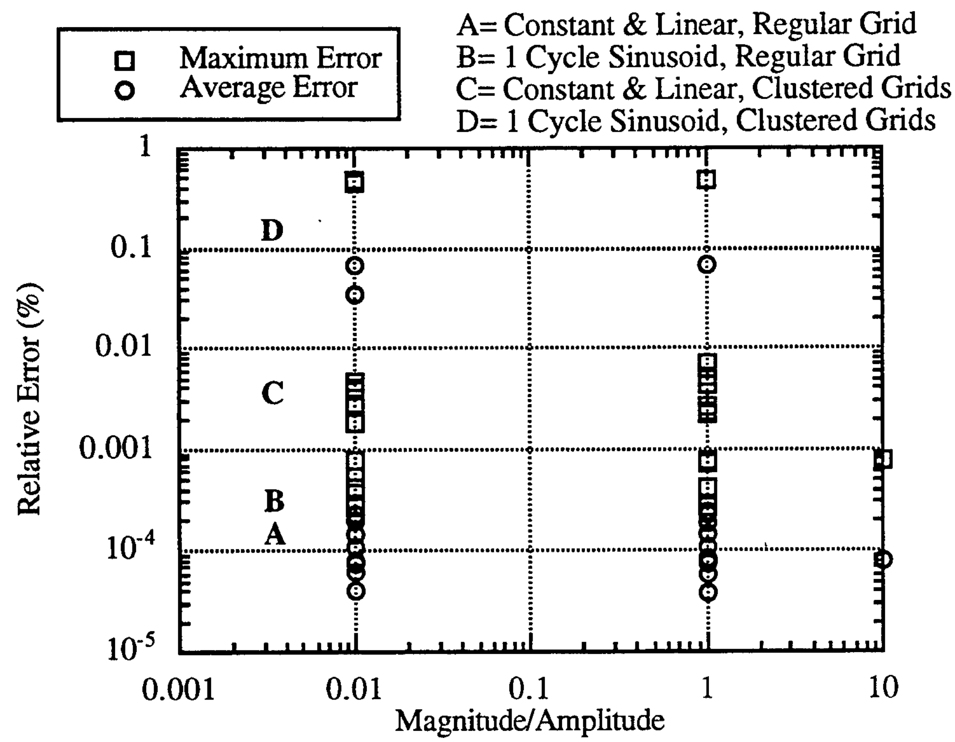


Figure 11-45. Variation of Error for Test Set One Based Upon the Order of Magnitude of the Function

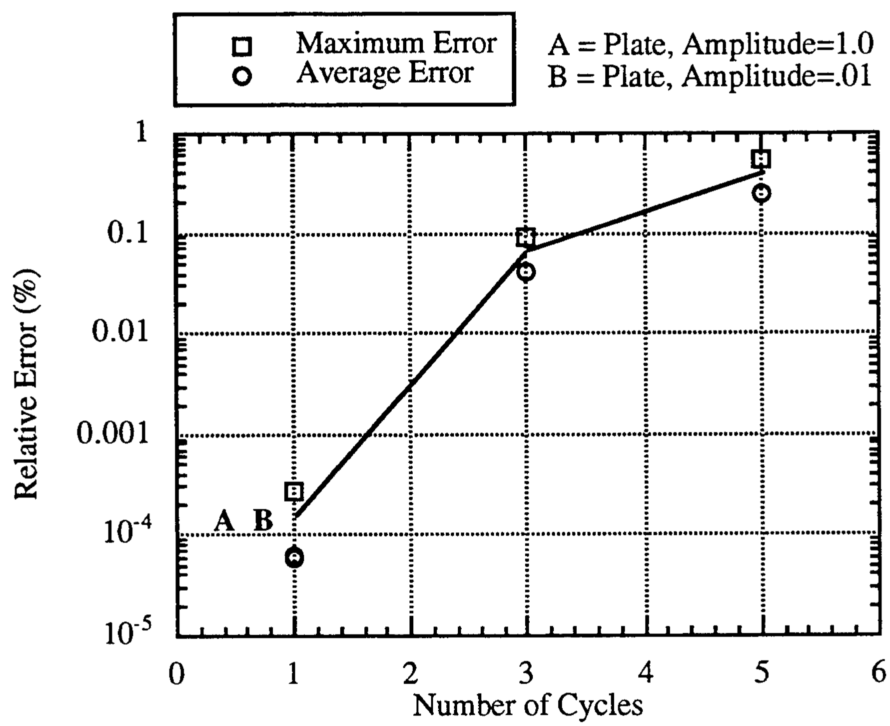
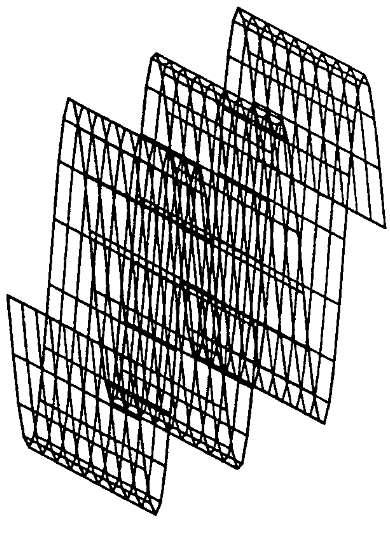
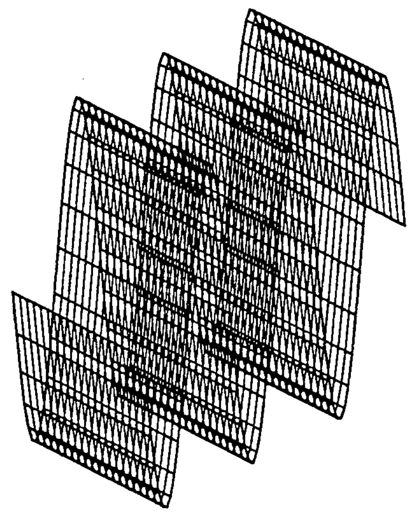


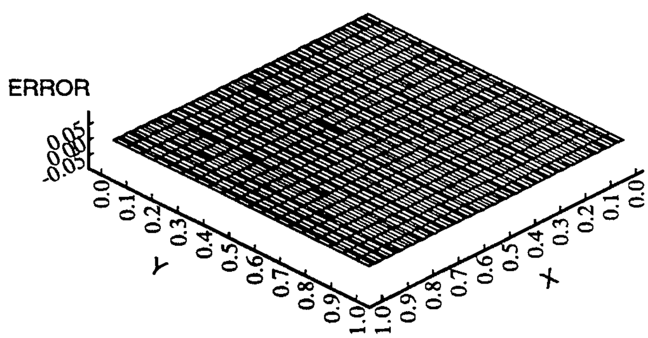
Figure 11-46. Variation of Error for Test Set One Based Upon the Number of Sinusoidal Cycles on the Surface Using Finite-Plate Splines



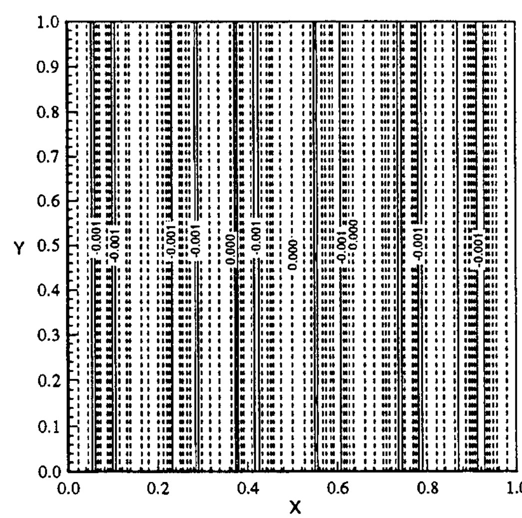
a) Original Function



b) Interpolated Function



c) Orthogonal View of Error



d) Error Contours

Figure 11-47. Example of Oscillations Induced by the Finite-Plate Spline Method (Test 1p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2 (X-axis and Y-axis have been expanded for visibility)

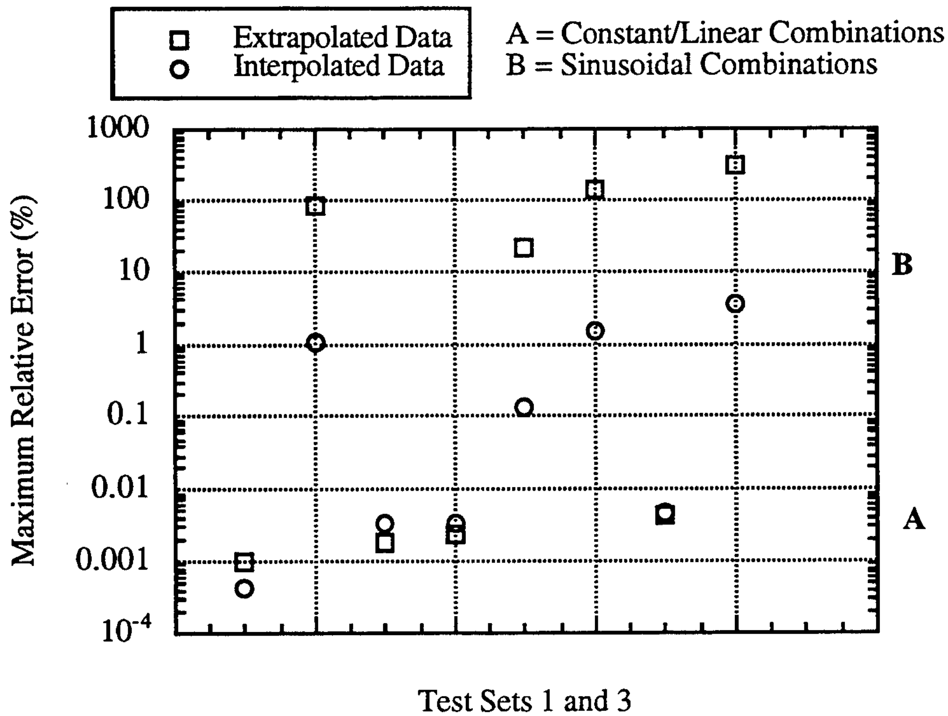


Figure 11-48. Variation of Error for Interpolation (Test Set 1) and Extrapolation (Test Set 3) Using Finite-Plate Splines

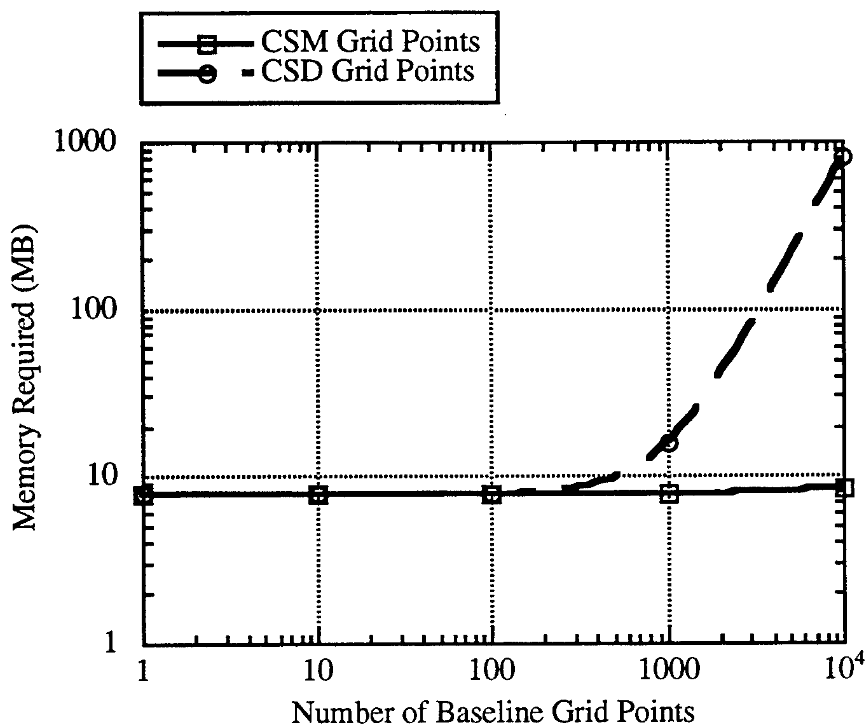


Figure 11-49. CPU Memory Requirements for the Finite Plate Method Implemented with a Maximum of 60 Quadrilateral Elements in the Virtual Mesh

Table 11.22 Statistical Summary of Test Set 1 using Finite-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	3.77E-05	7.53E-04	5	10	4.17E-06	8.35E-05	8.79E-06	22.655
test1b	2.11E-05	4.21E-04	5	10	1.97E-06	3.94E-05	4.45E-06	21.22563
test1c	5.47E-06	2.78E-04	5	10	1.20E-06	6.08E-05	2.09E-06	20.99726
test1d	2.60E-04	5.20E-03	44	5	1.21E-05	2.42E-04	3.28E-05	84.67223
test1e	1.35E-04	2.71E-03	32	2	7.43E-06	1.49E-04	1.90E-05	84.79054
test1f	9.74E-03	0.4881	23	20	1.39E-03	6.96E-02	3.32E-03	84.10423
test1g	2.11E-04	4.22E-03	9	20	9.73E-06	1.95E-04	1.08E-04	85.48438
test1h	1.08E-04	2.17E-03	8	19	3.86E-06	7.72E-05	1.23E-05	85.1853
test1i	9.74E-03	0.4882	28	1	1.39E-03	6.96E-02	2.46E-03	85.12653
test1j	3.49E-04	6.98E-03	26	2	1.12E-05	2.24E-04	8.40E-05	84.4776
test1k	1.17E-04	2.34E-03	25	4	5.55E-06	1.11E-04	1.66E-05	83.27924
test1l	9.71E-03	0.4889	13	1	1.32E-03	6.64E-02	2.84E-03	83.29193
test1m	3.78E-04	7.55E-04	5	10	4.14E-05	8.27E-05	2.99E-04	19.41486
test1n	7.64E-06	7.49E-04	5	10	8.15E-07	7.99E-05	3.01E-05	19.3714
test1o	1.09E-07	5.43E-04	5	10	1.24E-08	6.21E-05	3.03E-06	19.42792
test1p	1.82E-03	9.12E-02	41	5	8.11E-04	4.07E-02	1.25E-03	562.21747
test1q	1.07E-02	0.5355	89	20	4.80E-03	0.2402	7.29E-03	777.6712
test1s	1.82E-05	9.12E-02	30	1	8.11E-06	4.07E-02	1.25E-05	571.04498
test1t	1.07E-04	0.5355	89	1	4.80E-05	0.2402	1.79E-04	777.0238
test1a1	3.80E-07	7.61E-04	5	10	4.12E-08	8.24E-05	8.86E-08	22.285
test1a1	3.80E-07	7.61E-04	5	10	4.12E-08	8.24E-05	8.86E-08	22.7
test1b1	2.09E-07	4.17E-04	5	10	2.05E-08	4.11E-05	4.50E-08	21.12609
test1c1	1.36E-07	2.76E-04	5	10	3.06E-08	6.21E-05	5.20E-08	21.1282
test1d1	2.04E-06	4.08E-03	43	6	1.11E-07	2.22E-04	2.90E-07	83.64203
test1e1	1.87E-06	3.73E-03	32	2	7.41E-08	1.48E-04	1.92E-07	83.43233
test1f1	9.18E-05	0.4592	47	1	6.93E-06	3.47E-02	1.43E-05	83.75861
test1g1	1.93E-06	3.87E-03	9	11	1.03E-07	2.07E-04	5.29E-07	83.80688
test1h1	9.41E-07	1.88E-03	8	19	3.88E-08	7.75E-05	1.23E-07	83.93939
test1i1	9.74E-05	0.4882	28	1	1.39E-05	6.96E-02	2.46E-05	84.28174
test1j1	2.31E-06	4.61E-03	24	3	1.13E-07	2.26E-04	8.36E-07	83.49365
test1k1	1.36E-06	2.72E-03	18	2	5.41E-08	1.08E-04	1.60E-07	83.34625
test1l1	9.71E-05	0.4889	13	1	1.32E-05	6.64E-02	2.84E-05	82.69879
test1a2	3.77E-05	7.53E-04	5	10	4.17E-06	8.35E-05	8.79E-06	22.17
test1b2	2.12E-05	4.25E-04	5	10	2.11E-06	4.22E-05	4.67E-06	20.80391
test1c2	7.23E-02	3.67	5	5	4.43E-02	2.248	6.68E-02	20.565
test1d2	2.60E-04	5.20E-03	44	5	1.21E-05	2.42E-04	2.11E-03	82.4177
test1e2	1.60E-04	3.19E-03	41	8	9.44E-06	1.89E-04	7.13E-05	82.27765
test1f2	0.1067	5.334	33	11	3.90E-02	1.948	7.55E-02	82.21625
test1g2	2.11E-04	4.22E-03	9	20	9.73E-06	1.95E-04	2.39E-03	83.15231
test1h2	2.36E-04	4.72E-03	7	17	7.93E-06	1.59E-04	7.92E-05	82.95551
test1i2	0.1055	5.277	18	10	3.89E-02	1.946	4.36E-02	83.03882
test1j2	3.49E-04	6.98E-03	26	2	1.12E-05	2.24E-04	1.38E-03	82.35193
test1k2	1.40E-04	2.79E-03	25	4	8.53E-06	1.71E-04	4.93E-05	82.14563
test1l2	0.1066	5.345	18	6	4.42E-02	2.218	6.89E-02	82.11945
test1m2	3.78E-04	7.55E-04	5	10	4.14E-05	8.27E-05	6.92E-03	19.40344
test1n2	7.65E-06	7.50E-04	5	10	8.38E-07	8.22E-05	6.96E-04	19.27002
test1o2	1.18E-07	5.92E-04	6	1	1.36E-08	6.81E-05	6.99E-05	19.31665
test1p2	1.975	99.07	69	6	1.209	60.67	1.798	557.70251
test1a12	3.80E-07	7.61E-04	5	10	4.12E-08	8.24E-05	0.1807	18.91565
test1b12	2.12E-07	4.24E-04	5	10	2.07E-08	4.15E-05	1.82E-02	18.97412
test1c12	1.81E-03	3.67	5	5	1.11E-03	2.248	2.47E-03	18.81238
test1d12	2.04E-06	4.08E-03	43	6	1.11E-07	2.22E-04	7.82E-05	80.99048
test1e12	1.48E-06	2.97E-03	38	4	9.39E-08	1.88E-04	2.49E-06	80.99768
test1f12	1.06E-03	5.336	9	2	4.79E-04	2.404	7.22E-04	81.50488
test1g12	1.93E-06	3.87E-03	9	11	1.03E-07	2.07E-04	2.29E-05	82.01184
test1h12	1.74E-06	3.49E-03	6	20	7.49E-08	1.50E-04	7.56E-07	81.99829
test1i12	1.06E-03	5.279	18	10	3.89E-04	1.946	4.36E-04	82.01489
test1j12	2.31E-06	4.61E-03	24	3	1.13E-07	2.26E-04	1.38E-05	81.61572
test1k12	1.91E-06	3.83E-03	25	19	8.80E-08	1.76E-04	5.00E-07	81.42432
test1l12	1.07E-03	5.344	18	6	4.42E-04	2.218	6.89E-04	81.24243
test1A	4.23E-05	4.23E-04	5	10	3.93E-06	3.93E-05	8.50E-06	22.3
test1B	7.23E-02	1.037	5	5	4.43E-02	0.6353	6.68E-02	20.8225
test1C	3.44E-04	3.44E-03	25	2	1.73E-05	1.73E-04	2.11E-03	83.57156
test1D	3.23E-04	3.23E-03	25	2	1.63E-05	1.63E-04	8.15E-05	83.37236
test1E	9.71E-03	0.139	38	1	1.32E-03	1.89E-02	2.84E-03	84.20889
test1F	0.1066	1.524	18	6	4.42E-02	0.6326	6.89E-02	83.92169
test1G	9.63E-05	4.72E-03	24	17	5.34E-06	2.62E-04	2.18E-03	82.8642

Table 11.23 Statistical Summary of Test Set 2 using Finite-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test2A	4.77E-07	2.38E-05	6	4	1.55E-07	7.75E-06	2.39E-07	12.395
test2B	8.95E-07	4.47E-05	3	6	2.65E-07	1.33E-05	4.61E-07	10.98187
test2C	6.86E-07	3.43E-05	3	3	2.24E-07	1.12E-05	3.65E-07	10.73422
test2D	2.52E-06	4.20E-05	5	3	4.48E-07	7.47E-06	9.04E-07	10.57351
test2E	0.4204	21.34	8	8	0.103	5.231	0.1767	10.49516
test2F	1.07E-06	5.36E-05	21	10	1.50E-07	7.51E-06	2.60E-07	63.3125
test2G	1.33E-06	6.65E-05	25	3	2.93E-07	1.47E-05	4.41E-07	61.99458
test2H	1.26E-06	6.29E-05	23	10	2.93E-07	1.46E-05	4.36E-07	61.72594
test2I	3.09E-06	5.15E-05	23	10	4.41E-07	7.35E-06	7.76E-07	62.61963
test2L	5.58E-09	5.58E-05	23	10	5.50E-10	5.50E-06	1.56E-08	61.40564
test2M	2.82E-09	2.82E-05	9	36	4.66E-10	4.66E-06	8.67E-10	61.56732
test2N	4.51E-04	22.7	32	30	8.01E-05	4.035	1.32E-04	61.59891

Table 11.24 Statistical Summary of Test Set 3 using Finite-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test3A	5.18E-05	1.04E-03	2	10	4.44E-06	8.88E-05	9.64E-06	22.58
test3B	3.001	86.13	8	1	0.5506	15.8	0.9938	21.10781
test3C	9.13E-05	1.83E-03	26	5	1.03E-05	2.07E-04	3.14E-02	83.59679
test3D	1.17E-04	2.33E-03	26	2	7.74E-06	1.55E-04	9.95E-04	83.60129
test3E	0.7582	21.7	1	6	0.1962	5.616	0.4878	83.45741
test3F	4.697	134.3	26	1	1.074	30.7	1.325	82.8118
test3G	4.29E-05	4.20E-03	26	2	4.55E-06	4.46E-04	4.19E-02	82.64542
test3H	4.697	311.7	26	1	1.074	71.23	1.325	82.66898
test3A3	9.54E-07	4.77E-05	4	1	1.90E-07	9.48E-06	3.29E-07	12.23
test3B3	8.05E-07	4.03E-05	3	3	2.78E-07	1.39E-05	4.37E-07	11.00344
test3C3	8.50E-07	4.25E-05	4	2	2.78E-07	1.39E-05	4.40E-07	10.815
test3D3	4.14E-06	6.91E-05	3	2	5.39E-07	8.98E-06	1.09E-06	10.5832
test3E3	0.463	23.51	10	9	9.51E-02	4.828	0.1621	10.44476
test3F3	2.38E-06	1.19E-04	2	2	3.02E-07	1.51E-05	5.12E-07	62.7775
test3G3	2.88E-06	1.44E-04	9	3	3.74E-07	1.87E-05	6.07E-07	61.66628
test3H3	2.85E-06	1.43E-04	2	2	3.64E-07	1.82E-05	5.91E-07	61.40768
test3I3	7.01E-06	1.17E-04	2	2	6.89E-07	1.15E-05	1.19E-06	61.31207
test3J3	0.4808	24.21	50	38	0.113	5.69	0.1604	61.13815
test3K3	1.40E-08	1.40E-04	9	3	1.54E-09	1.54E-05	3.21E-03	61.22037
test3L3	1.21E-08	1.21E-04	3	4	1.02E-09	1.02E-05	6.42E-05	60.8923
test3M3	8.37E-09	8.37E-05	2	45	8.55E-10	8.55E-06	1.28E-06	60.89447
test3N3	4.81E-04	24.21	50	38	1.13E-04	5.69	1.60E-04	60.89655

Table 11.25 Statistical Summary of Test Set 5 using Finite-Plate Splines

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test5a	1.92E-07	1.92E-05	9	1	0.00E+00	0.00E+00	1.02E-07	0.625
test5b	1.87E-07	1.87E-05	6	1	0.00E+00	0.00E+00	6.89E-08	1.155
test5c	1.51E-07	1.51E-05	6	1	0.00E+00	0.00E+00	6.62E-08	1.31125
test5d	3.82E-05	3.82E-03	100	1	0.00E+00	0.00E+00	1.71E-05	1.4375
test5e	1.15E-03	0.1152	50	1	1.14E-06	1.14E-04	6.97E-04	1.16062
test5f	6.75E-03	0.6745	47	1	1.20E-05	1.20E-03	4.05E-03	0.99906
test5g	9.89E-04	9.89E-02	1	1	5.36E-08	5.36E-06	1.82E-04	11.41031
test5h	1.38E-03	0.1377	246	1	7.56E-07	7.56E-05	7.13E-04	11.15156
test5i	6.71E-03	0.6711	129	1	3.64E-06	3.64E-04	4.11E-03	11.03109
test5j	9.89E-04	0.9894	1	1	3.10E-10	3.10E-07	1.38E-03	2.57547
test5k	1.27E-08	1.27E-05	6	1	7.17E-10	7.17E-07	4.59E-04	2.58086
test5l	1.66E-08	1.66E-05	6	1	4.84E-10	4.84E-07	1.53E-04	2.58625
test5m	3.81E-06	3.81E-03	100	1	1.30E-09	1.30E-06	1.55E-05	0.27703
test5n	1.15E-04	0.1151	50	1	1.04E-07	1.04E-04	6.98E-05	0.25348
test5o	6.75E-04	0.6745	47	1	1.20E-06	1.20E-03	4.05E-04	0.21836
test5p	9.90E-05	9.90E-02	1	1	8.90E-10	8.90E-07	1.82E-05	10.77246
test5k	3.81E-06	3.81E-03	1	1	7.17E-10	7.17E-07	6.09E-06	2.72566
test5r	6.71E-04	0.6711	129	1	1.49E-09	1.49E-06	4.11E-04	10.7702
test5a1	9.90E-05	3.30E-03	1	1	0.00E+00	0.00E+00	1.38E-04	2.76801
test5b1	3.99E-07	1.33E-05	6	1	0.00E+00	0.00E+00	4.59E-05	2.77074
test5c1	2.69E-07	8.95E-06	6	1	0.00E+00	0.00E+00	1.53E-05	2.77348
test5d1	6.63E-05	2.21E-03	100	1	0.00E+00	0.00E+00	2.77E-05	0.10324
test5e1	2.31E-03	7.71E-02	50	1	2.16E-06	7.21E-05	1.40E-03	0.06816
test5f1	1.35E-02	0.4496	47	1	2.61E-05	8.70E-04	8.10E-03	0.06301
test5g1	1.99E-03	6.63E-02	1	1	2.91E-08	9.69E-07	3.64E-04	10.64707
test5h1	2.81E-03	9.38E-02	246	1	5.38E-06	1.79E-04	1.42E-03	10.5285
test5i1	1.34E-02	0.4467	129	1	3.46E-06	1.15E-04	8.21E-03	10.54875
test5j1	1.99E-03	0.663	1	1	1.76E-10	5.88E-08	2.74E-03	2.93037
test5k1	2.94E-08	9.79E-06	9	1	7.64E-10	2.55E-07	9.15E-04	2.93173
test5l1	2.98E-08	9.95E-06	6	1	3.74E-09	1.25E-06	3.05E-04	2.93312
test5m1	6.66E-06	2.22E-03	100	1	1.19E-08	3.97E-06	3.08E-05	0.06477
test5n1	2.31E-04	7.71E-02	50	1	2.40E-07	8.00E-05	1.40E-04	0.07231
test5o1	1.35E-03	0.4496	47	1	2.60E-06	8.68E-04	8.10E-04	0.07985
test5p1	1.99E-04	6.63E-02	1	1	9.01E-09	3.01E-06	3.64E-05	10.49214
test5k1	6.66E-06	2.22E-03	1	1	7.64E-10	2.55E-07	1.22E-05	2.99696
test5r1	1.34E-03	0.4467	129	1	1.19E-08	3.97E-06	8.21E-04	10.45093

11.6 Inverse Isoparametric Method

The inverse isoparametric method was implemented by Dr. R. Fithen [15] of Wright Laboratory, as discussed previously in Chapter 9. This methodology is a two-dimensional application, so that the testing was constrained by the following: two-dimensional surfaces (plates), regular grids, no extrapolation, and no beam element implementation. Therefore, the only mathematical tests which could be performed were limited to portions of test sets 1 and 4. The following observations were made using a limited test set. Following any extension to three-dimensions, these observations may no longer be valid.

Overall Accuracy – The overall accuracy of the code for the test cases which were examined was very good. The maximum error encountered was approximately 5%. Most of the errors remained much lower than 1%, as shown in Tables 11.26 and 11.27. The method had no problems running any of the two-dimensional test cases, within the scope previously detailed.

Grid Spacing Sensitivity – Figure 11-50 indicates that the inverse isoparametric method is not sensitive to grid spacing. Where the function was transferred to the identical grid, the errors were identically 0.0 and were not plotted.

Additionally, the interpolated data are shown to be constant along the direction for which the function is also constant. This is visually depicted in Figure 11-51 which is a typical interpolation error plot by the inverse isoparametric method. There are some oscillations in the direction of the function. The negative impact of larger oscillations can be felt for interpolation of grid deflections from the structural to the aerodynamic grid. Oscillations in the updated surface grid of a wing can result in non-physical pressure distributions, and can cause flow separation bubbles or transition to turbulent flow where laminar flow actually occurs. The errors for the inverse isoparametric method fluctuate less than 1% of the overall magnitude for these runs, and thus the oscillations should be large enough to impact the interpolation in the manner which was just discussed.

Directional Bias – Figure 11-50 includes the streamwise (test1a – 11) and spanwise (test1a2 – 12) functions examined in test set 1. These data are virtually identical for both directions, indicating little or no sensitivity to direction of the function. This is important since the orientation of the surface to be interfaced is not known, and the dominant function (the data which change the most) can be located in either direction on the surface.

Magnitude/Amplitude Sensitivity – IIM is not sensitive to the amplitude of sinusoidal functions. This trend is consistent for function type, direction of the function and grid. This trend indicates that the percentage error (refer to Table 11.26) does not change with an increase or decrease of function amplitude or magnitude.

Sensitivity to Frequency (Higher Oscillations) – The inverse isoparametric method is relatively insensitive to the frequency of the function. As shown in Figure 11-52, the errors are fairly consistent as the sinusoidal function is increased over the surface.

The interpolation result for a three-cycle sinusoid is plotted in Figure 11-53. The error trends are similar to those for a one-cycle sinusoid, as plotted in Figure 11-51. Some very small oscillations which are less than 1% of the overall amplitude of the function are seen in the direction of the varying function. Some degradation in capturing the overall peak amplitudes are seen in Figures 11-53 c) and d), but an overall smooth function is generated, as depicted in Figure 11-53 b). Figure 11-53 d) shows that the errors are still impervious to the direction where the function remains constant.

Diminishing Variation – Figure 11-54 shows that the error does not vary with increasing fineness, indicating that IIM is consistent for each function.

Algorithm CPU Memory and Time Requirements – The average CPU time requirement is approximately 1 second (refer to the last column of Tables 11.26 and 11.27), with the exception of the test cases which had very large grids which required almost 60 seconds (1 minute) of CPU time.

The algorithm CPU memory requirement can be computed using the algorithm :

$$\text{CPU SIZE (MBytes)} = 0.251 + 7.5958\text{x}10^{-5}\text{KGS} + 3.2121\text{x}10^{-12}\text{KGS} + 8.7909\text{x}10^{-5}\text{UKS} + 9.4938\text{x}10^{-12}\text{UKS}$$

where KGS is the total number of input grid points and UKS is the total number of grid points for the grid to be interpolated to. The impact of increasing grid sizes can be seen in Figure 11-55.

Single Precision – The inverse isoparametric method was implemented in double precision. Because of the nature of the mathematical algorithm, single precision was not attempted.

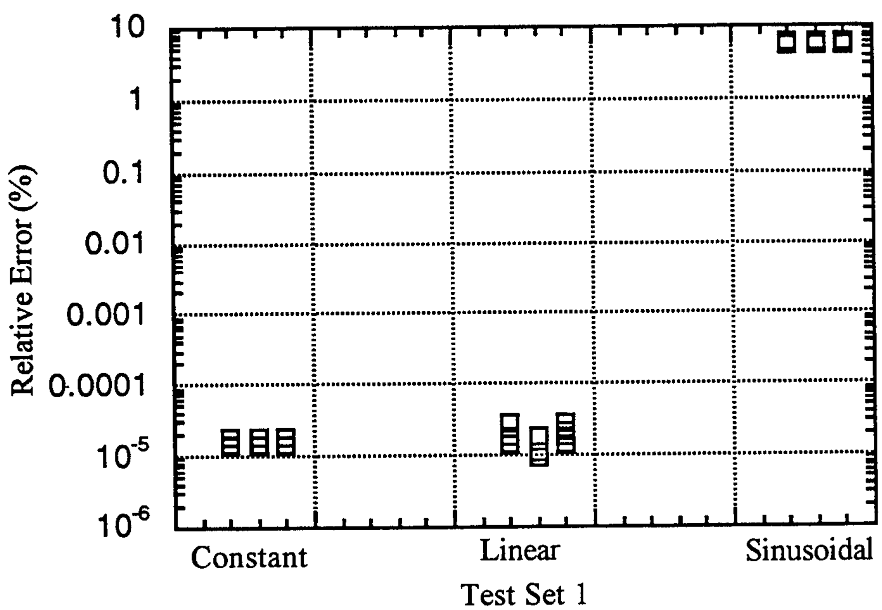
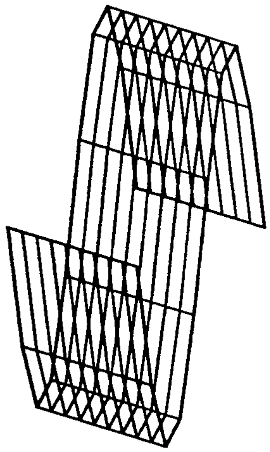
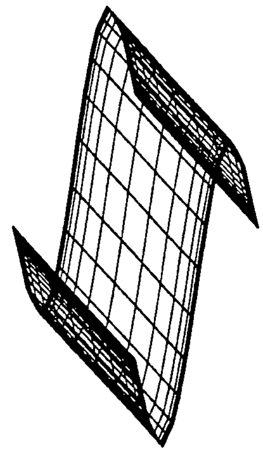
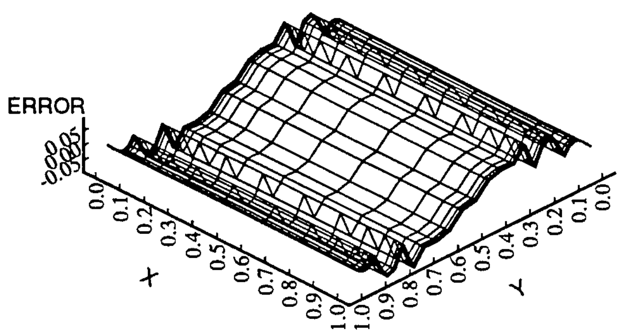


Figure 11-50. Variation of Error for Test Set One Based Upon Function Type Using the Inverse Isoparametric Method (Regular Grid Errors are all 0.0 and are not Plotted)

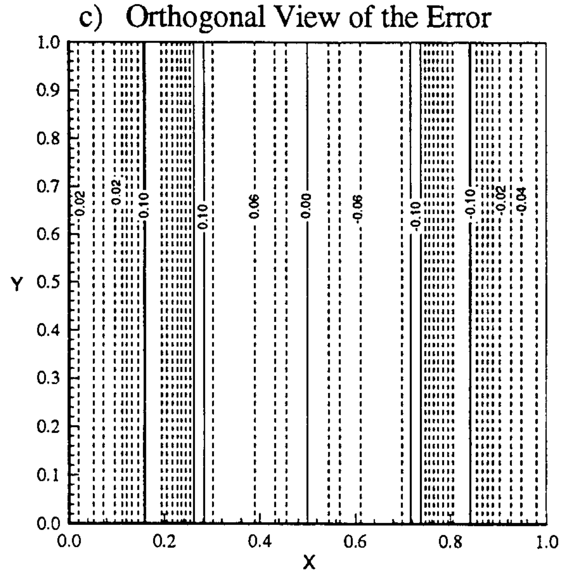




a) Original Function b) Interpolated Function



c) Orthogonal View of the Error



d) Error Contours

Figure 11-51. Example of Oscillations Induced by the Inverse Isoparametric Method (Test l) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2

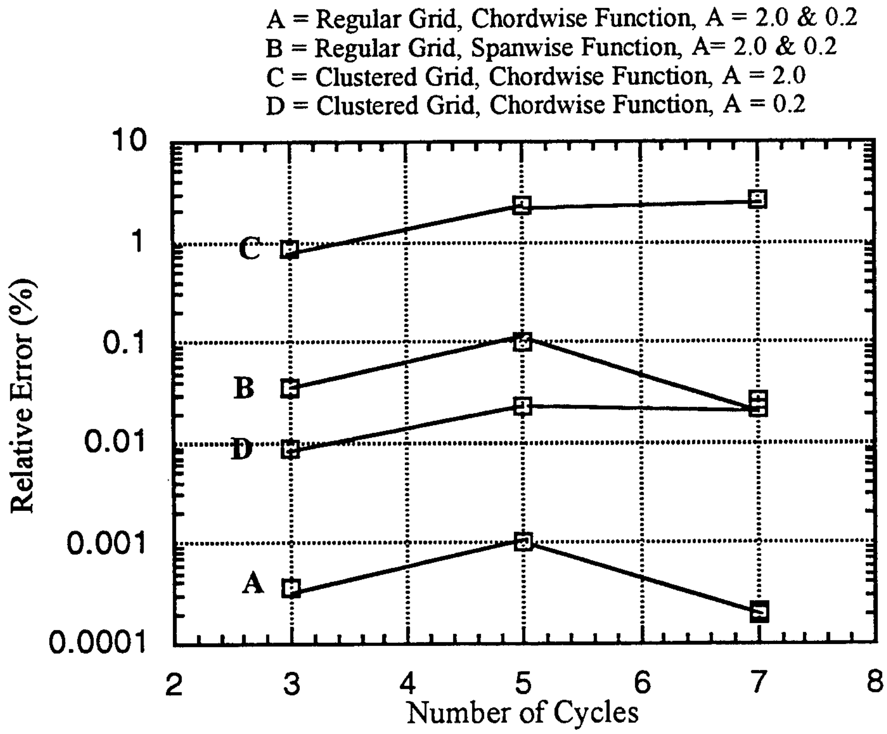
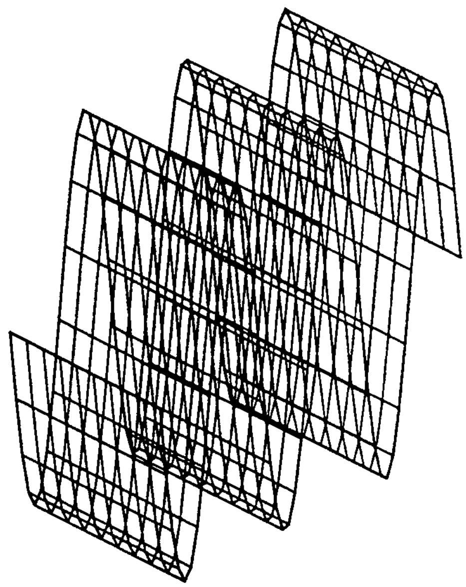
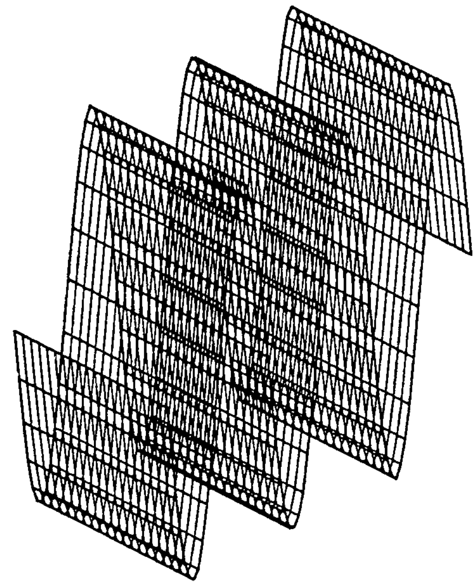


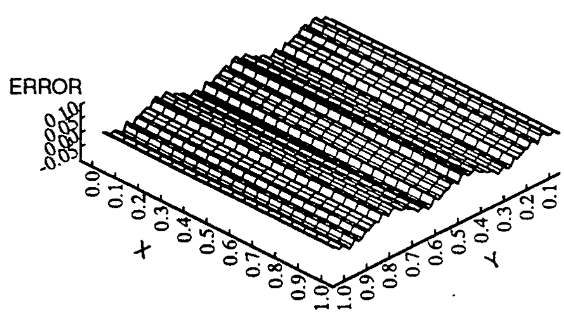
Figure 11-52. Variation of Error for Test Set One Based Upon Number of Sinusoidal Cycles Using the Inverse Isoparametric Method



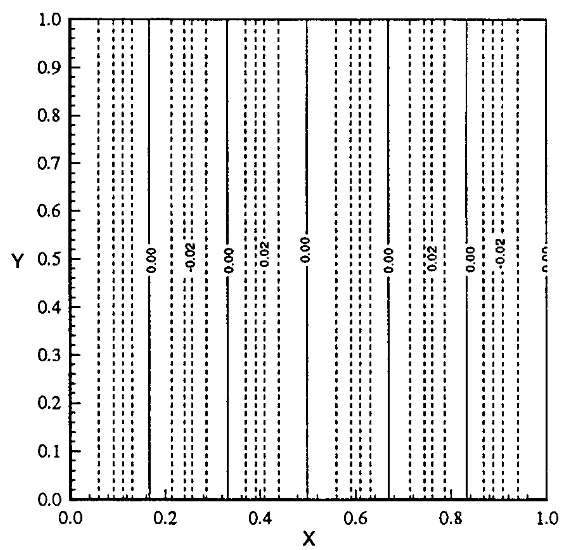
a) Original Function



b) Interpolated Function



c) Orthogonal View of Error



d) Error Contours

Figure 11-53. Example of Oscillations Induced by the Inverse Isoparametric Method (Test p) for a Three-Cycle Sinusoidal Function at a Peak-to-Peak Amplitude of 2 (X-axis and Y-axis have been expanded for visibility)

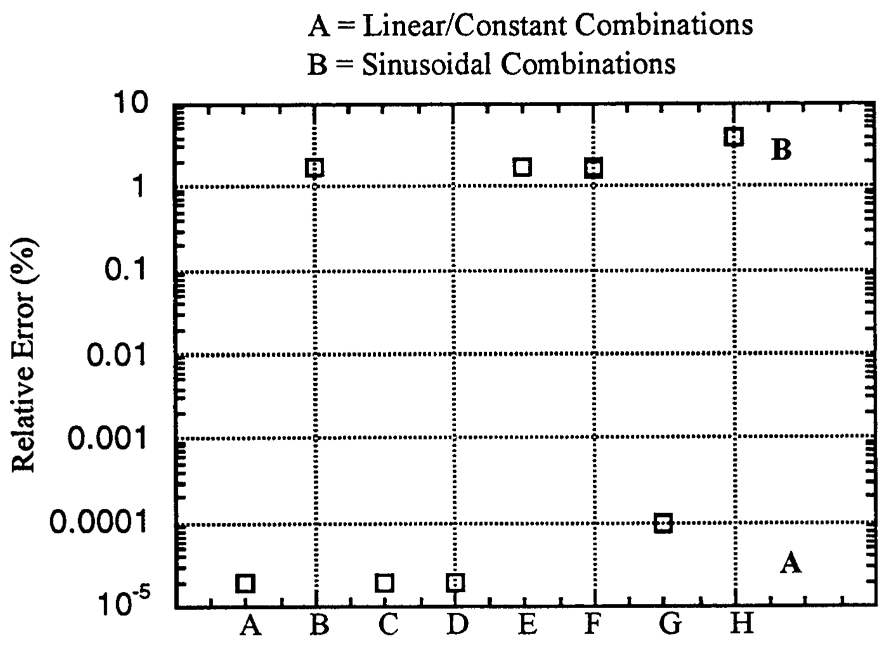


Figure 11-54. Variation of Error for Test Set 4 Based Upon Number of Sinusoidal Cycles Using the Inverse Isoparametric Method

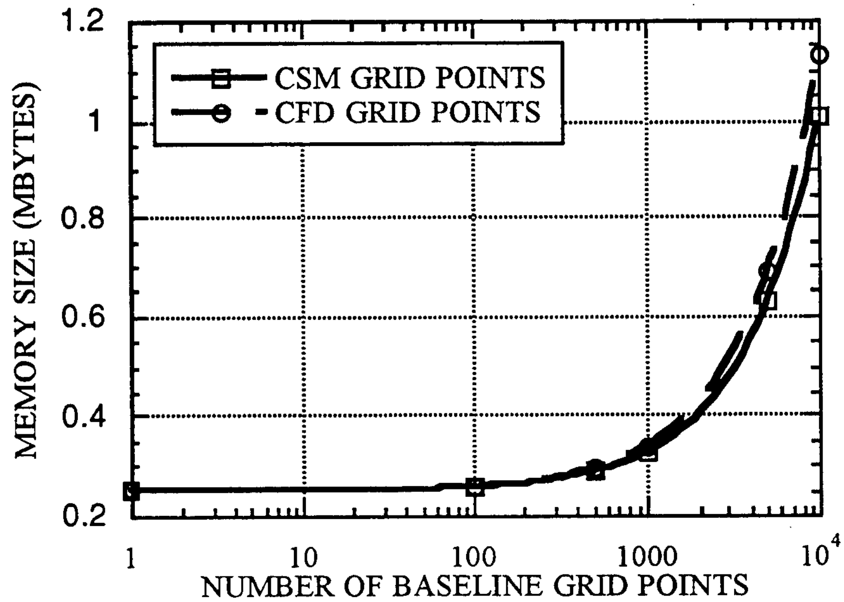


Figure 11-55. CPU Memory Requirements for the Inverse Isoparametric Method

Table 11.26 Statistical Summary of Test Set 1 using Inverse Isoparametric Mapping

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.06
test1b	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.05
test1c	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.05
test1d	9.54E-07	1.91E-05	31	14	0.00E+00	0.00E+00	2.69E-07	0.48
test1e	1.43E-06	2.86E-05	27	1	0.00E+00	0.00E+00	6.58E-07	0.47
test1f	0.1182	5.922	47	7	0.00E+00	0.00E+00	4.37E-02	0.47
test1g	9.54E-07	1.91E-05	7	9	0.00E+00	0.00E+00	1.38E-03	0.54
test1h	9.54E-07	1.91E-05	2	1	0.00E+00	0.00E+00	4.38E-05	0.54
test1i	0.1181	5.922	4	9	0.00E+00	0.00E+00	7.11E-02	0.54
test1j	9.54E-07	1.91E-05	48	3	0.00E+00	0.00E+00	2.25E-03	0.5
test1k	1.43E-06	2.86E-05	13	4	0.00E+00	0.00E+00	7.12E-05	0.5
test1l	0.1159	5.837	30	3	0.00E+00	0.00E+00	5.83E-02	0.51
test1m	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.85E-03	0.05
test1n	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.88E-04	0.05
test1o	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.91E-05	0.05
test1p	3.56E-02	1.786	6	2	0.00E+00	0.00E+00	2.44E-02	1.06
test1q	0.1016	5.08	95	2	0.00E+00	0.00E+00	6.76E-02	1.52
test1r	2.14E-02	1.071	178	1	0.00E+00	0.00E+00	1.45E-02	4.73
test1s	3.56E-04	1.786	6	2	0.00E+00	0.00E+00	4.59E-04	1.08
test1t	1.02E-03	5.08	6	1	0.00E+00	0.00E+00	6.76E-04	1.54
test1u	2.14E-04	1.071	178	2	0.00E+00	0.00E+00	1.45E-04	4.7
test1a1	0.00E+00	0.00E+00	178	2	0.00E+00	0.00E+00	1.46E-05	0.06
test1b1	0.00E+00	0.00E+00	178	2	0.00E+00	0.00E+00	1.47E-06	0.05
test1c1	0.00E+00	0.00E+00	178	2	0.00E+00	0.00E+00	1.47E-07	0.05
test1d1	7.45E-09	1.49E-05	45	4	0.00E+00	0.00E+00	5.71E-09	0.49
test1e1	9.31E-09	1.86E-05	42	7	0.00E+00	0.00E+00	3.24E-09	0.47
test1f1	1.16E-03	5.803	15	2	0.00E+00	0.00E+00	7.94E-04	0.49
test1g1	7.45E-09	1.49E-05	7	9	0.00E+00	0.00E+00	2.51E-05	0.54
test1h1	5.59E-09	1.12E-05	12	3	0.00E+00	0.00E+00	7.95E-07	0.54
test1i1	1.18E-03	5.922	4	7	0.00E+00	0.00E+00	7.11E-04	0.54
test1j1	7.45E-09	1.49E-05	22	3	0.00E+00	0.00E+00	2.25E-05	0.52
test1k1	1.12E-08	2.24E-05	23	2	0.00E+00	0.00E+00	7.12E-07	0.51
test1l1	1.16E-03	5.837	21	1	0.00E+00	0.00E+00	5.83E-04	0.49
test1a2	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.05
test1b2	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.06
test1c2	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.05
test1d2	9.54E-07	1.91E-05	31	14	0.00E+00	0.00E+00	2.69E-07	0.47
test1e2	1.43E-06	2.86E-05	47	13	0.00E+00	0.00E+00	7.37E-07	0.48
test1f2	0.1115	5.574	38	19	0.00E+00	0.00E+00	4.34E-02	0.48
test1g2	9.54E-07	1.91E-05	7	9	0.00E+00	0.00E+00	1.37E-03	0.53
test1h2	9.54E-07	1.91E-05	2	2	0.00E+00	0.00E+00	4.35E-05	0.54
test1i2	0.1115	5.574	10	2	0.00E+00	0.00E+00	6.58E-02	0.53
test1j2	9.54E-07	1.91E-05	48	3	0.00E+00	0.00E+00	2.08E-03	0.5
test1k2	9.54E-07	1.91E-05	11	2	0.00E+00	0.00E+00	6.59E-05	0.53
test1l2	0.1099	5.509	2	12	0.00E+00	0.00E+00	6.40E-02	0.52
test1m2	0.00E+00	0.00E+00	2	12	0.00E+00	0.00E+00	6.44E-03	0.05
test1n2	0.00E+00	0.00E+00	2	12	0.00E+00	0.00E+00	6.47E-04	0.05
test1o2	0.00E+00	0.00E+00	2	12	0.00E+00	0.00E+00	6.50E-05	0.05
test1p2	0.8539	42.84	3	2	0.00E+00	0.00E+00	0.6327	1.07
test1q2	2.317	116.3	1	2	0.00E+00	0.00E+00	1.487	1.53
test1r2	2.657	133.3	5	4	0.00E+00	0.00E+00	2.269	4.69
test1s2	8.54E-03	42.84	5	2	0.00E+00	0.00E+00	6.10E-02	1.09
test1t2	2.32E-02	116.3	90	2	0.00E+00	0.00E+00	1.49E-02	1.53
test1u2	2.66E-02	133.3	1	17	0.00E+00	0.00E+00	2.27E-02	4.76

Table 11.26 Statistical Summary of Test Set 1 using Inverse Isoparametric Mapping (Cont.)

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test1a12	0.00E+00	0.00E+00	1	17	0.00E+00	0.00E+00	2.28E-03	0.05
test1b12	0.00E+00	0.00E+00	1	17	0.00E+00	0.00E+00	2.29E-04	0.05
test1c12	0.00E+00	0.00E+00	1	17	0.00E+00	0.00E+00	2.30E-05	0.06
test1d12	7.45E-09	1.49E-05	45	4	0.00E+00	0.00E+00	7.29E-07	0.49
test1e12	7.45E-09	1.49E-05	20	3	0.00E+00	0.00E+00	2.33E-08	0.47
test1f12	1.11E-03	5.573	8	6	0.00E+00	0.00E+00	7.81E-04	0.49
test1g12	7.45E-09	1.49E-05	7	9	0.00E+00	0.00E+00	2.47E-05	0.54
test1h12	4.66E-09	9.31E-06	32	12	0.00E+00	0.00E+00	7.81E-07	0.54
test1i12	1.12E-03	5.574	1	2	0.00E+00	0.00E+00	6.58E-04	0.54
test1j12	7.45E-09	1.49E-05	22	3	0.00E+00	0.00E+00	2.08E-05	0.54
test1k12	7.45E-09	1.49E-05	11	3	0.00E+00	0.00E+00	6.59E-07	0.51
test1l12	1.10E-03	5.508	31	12	0.00E+00	0.00E+00	6.40E-04	0.5
test1bb2	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.05
test1cc2	0.00E+00	0.00E+00	0	0	0.00E+00	0.00E+00	0.00E+00	0.06
test1dd2	9.54E-07	1.91E-05	31	14	0.00E+00	0.00E+00	2.69E-07	0.47
test1ee2	1.43E-06	2.86E-05	27	1	0.00E+00	0.00E+00	6.58E-07	0.47
test1ff2	0.1182	5.922	47	7	0.00E+00	0.00E+00	4.37E-02	0.46
test1gg2	9.54E-07	1.91E-05	7	9	0.00E+00	0.00E+00	1.38E-03	0.53
test1hh2	9.54E-07	1.91E-05	2	1	0.00E+00	0.00E+00	4.38E-05	0.54
test1ii2	0.1181	5.922	4	9	0.00E+00	0.00E+00	7.11E-02	0.54
test1jj2	9.54E-07	1.91E-05	48	3	0.00E+00	0.00E+00	2.25E-03	0.52
test1kk2	1.43E-06	2.86E-05	13	4	0.00E+00	0.00E+00	7.12E-05	0.49
test1ll2	0.1159	5.837	30	3	0.00E+00	0.00E+00	5.83E-02	0.51
test1mm2	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.85E-03	0.04
test1nn2	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.88E-04	0.05
test1oo2	0.00E+00	0.00E+00	30	3	0.00E+00	0.00E+00	5.91E-05	0.05
test1pp2	3.56E-02	1.786	6	2	0.00E+00	0.00E+00	2.44E-02	1.07
test1qq2	0.1016	5.08	95	2	0.00E+00	0.00E+00	6.76E-02	1.53
test1rr2	2.14E-02	1.071	178	1	0.00E+00	0.00E+00	1.45E-02	4.73
test1ss2	3.56E-04	1.786	6	2	0.00E+00	0.00E+00	4.59E-04	1.09
test1tt2	1.02E-03	5.08	6	1	0.00E+00	0.00E+00	6.76E-04	1.55
test1uu2	1.96E-04	0.9806	11	2	0.00E+00	0.00E+00	1.45E-04	0.48

Table 11.27 Statistical Summary of Test Set 4 using Inverse Isoparametric Mapping

Test Case	Maximum Error				Average Error		Standard	CPU Time
	Absolute	Percentage	I Location	J Location	Absolute	Percentage	Deviation	(Seconds)
test4A2	1.00E-06	2.00E-05	5	1	1.26E-07	2.52E-06	1.26E-07	0.68
test4A3	1.00E-06	2.00E-05	4	1	1.22E-07	2.45E-06	1.22E-07	4.09
test4B2	5.89E-02	1.696	1	12	2.47E-02	0.7107	2.47E-02	8.74
test4B3	5.94E-02	1.707	7	23	2.52E-02	0.725	2.52E-02	12.06
test4C2	1.00E-06	2.00E-05	3	1	1.00E-07	2.00E-06	1.99E-04	21.66
test4C3	1.00E-06	2.00E-05	2	1	1.25E-07	2.50E-06	7.98E-07	53.48
test4D2	1.00E-06	2.00E-05	3	1	1.24E-07	2.47E-06	1.24E-07	98.66
test4D3	1.00E-06	2.00E-05	2	1	1.23E-07	2.47E-06	1.23E-07	130.55
test4E2	5.94E-02	1.707	188	4	1.79E-02	0.5146	1.79E-02	175.60001
test4E3	5.94E-02	1.705	305	9	1.80E-02	0.5161	1.80E-02	207.8
test4F2	5.73E-02	1.645	1	35	1.48E-02	0.4242	1.48E-02	253.53
test4F3	5.92E-02	1.702	47	69	1.49E-02	0.4296	1.49E-02	285.92001
test4G2	1.00E-06	9.80E-05	79	1	5.29E-08	5.18E-06	1.18E-04	331.03
test4G3	1.00E-06	9.80E-05	94	1	5.44E-08	5.33E-06	4.70E-07	363.04001
test4H2	5.73E-02	3.838	163	35	1.48E-02	0.9901	1.48E-02	408.12
test4H3	5.92E-02	3.979	42	69	1.49E-02	1.005	1.49E-02	440.54001

12. DESCRIPTION OF THE APPLICATIONS TEST CASES

It is necessary to examine the performance of these methods, used to analyze the analytical test cases of the two previous chapters, on real problems. This will ensure that the conclusions that have been derived from the analytical examination are valid. Several applications test cases have been investigated. These applications test cases are described in the following sections.

There are three types of data which are evaluated within the applications test cases. There are two methods which can be used in coupling (loosely or tightly) aerodynamic and structural methodologies. First, the mode shapes or influence coefficients can be interpolated from the structural grid to the aerodynamic grid. The structural equations of motion are then solved using the larger (usually) aerodynamic mesh. The advantage of this method is that the loads integration can occur directly on the aerodynamic mesh, without the introduction of numerical interpolation errors in the rapidly changing pressure data.

The other aeroelastic process to convert the aerodynamic pressures to the structural mesh points, solve the structural equations of motion, then interpolate the deflections back to the aerodynamic grid. This has the advantage that for most cases, the structural meshes are much smaller than the aerodynamic surface grids and are much quicker to solve.

For these applications test cases, data which could be used by either of the two processes was examined. Mode shape data was interpolated to provide deflection-like information, influence coefficients were interpolated for process 1, and loads were interpolated and integrated for the second process. The results of these test cases are described in Chapter 13.

12.1 AGARD 445 Wing

The first test case examined is the interpolation of five mode shapes of the AGARD 445 wing [16]. This test case represents one of the primary types of configurations, a lifting-surface, that is analyzed by higher-order tightly-coupled aeroelastic methods.

The wing structure is represented by a flat plate which extends from the wing leading to trailing edges and from the wing root to the wing tip. This case involves pure interpolation with no extrapolation. The wing is a lifting surface whose motion is dominated by the motion in the z (Cartesian) coordinate. The x and y Cartesian coordinate motions are neglected.

In this test case, the structural grid is a regularly spaced mesh, 11 nodes in the streamwise direction and 31 nodes in the spanwise direction, shown in Figure 12-1. The CFD grid encloses the actual wing surface, and is comprised of 219 streamwise (110 on upper and lower surfaces) and 21 spanwise nodes. The grid is clustered at the leading and trailing edges, as seen in Figure 12-2. In these two figures, the view is a perspective of the wing which correlates with the mode shape deflection plots described in the following paragraphs. The geometry of the wing is from left to right (leading to trailing edge) and from bottom to top (root to tip).

The first five modes of the wing are shown graphically in Figures 12-3 through 12-7. The a) plot of each figure is the contour of the mode shapes. Each major increment of deflection is delineated with a solid line, with minor contours as dashed lines.

The b) plot of each figure has the mode shape (times a unit increment) superimposed on the surface to demonstrate the unit deflection of the surface. In order to provide a good visual perspective, the z Cartesian coordinate (direction of the deflections) has been expanded with respect to the wing planform (x and y Cartesian coordinates).

For each plot, the wing root lies along the abscissa of each plot, with the leading edge located at 0.0 and the trailing edge at 1.8. The ordinate axis is the spanwise coordinate, with wing root at 0.0 and the wing tip at 2.5. The undeflected wing is outlined by the surface boundaries and is shown in perspective at the same scale as the deflections.

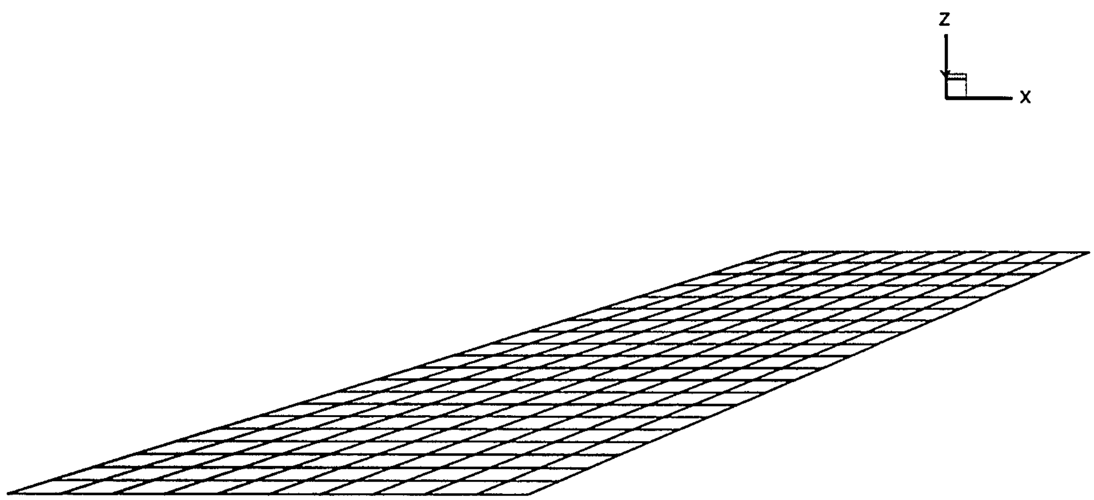


Figure 12-1. AGARD 445 Wing Structural Grid (Undeflected)

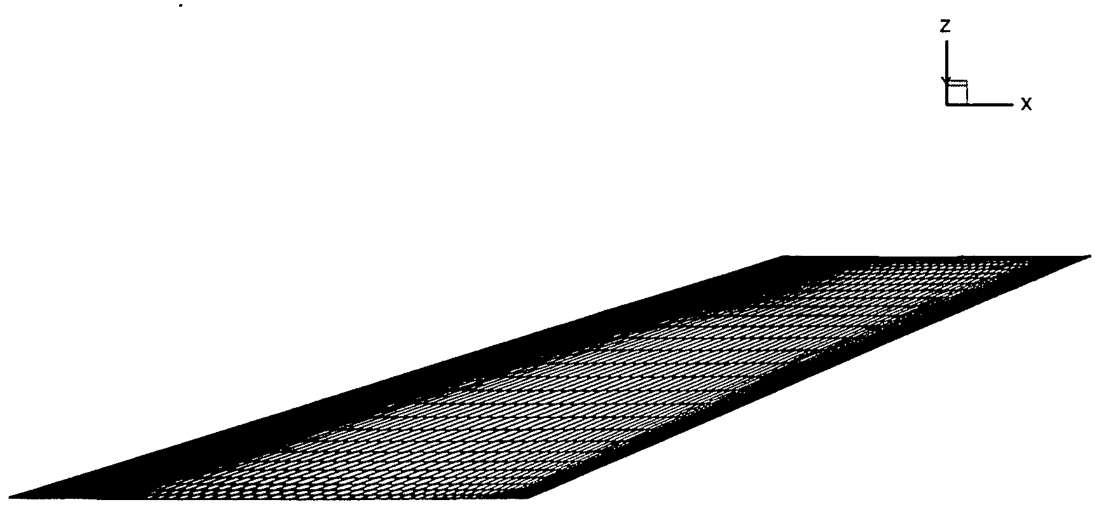
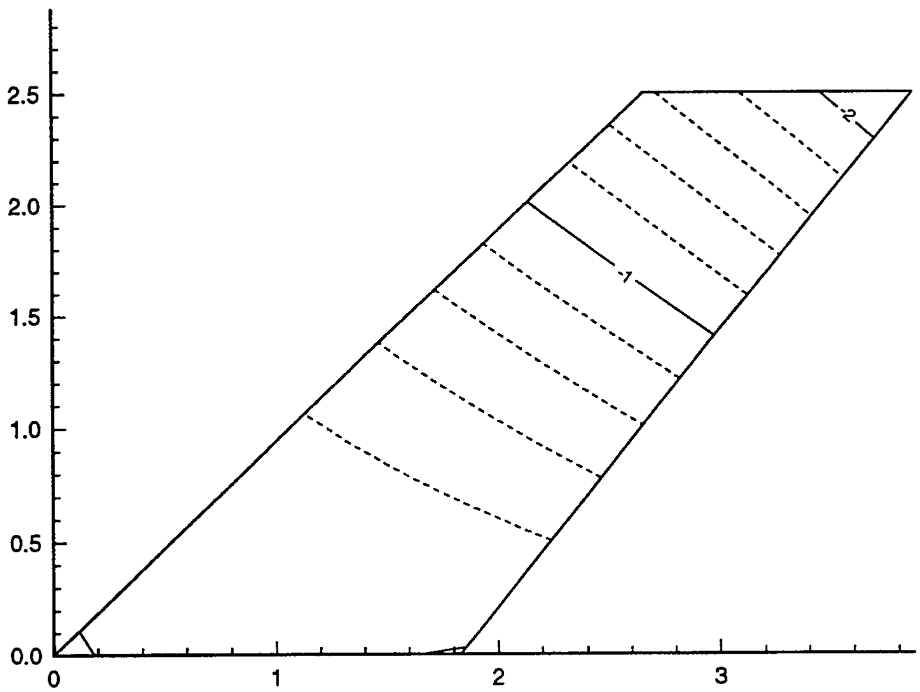
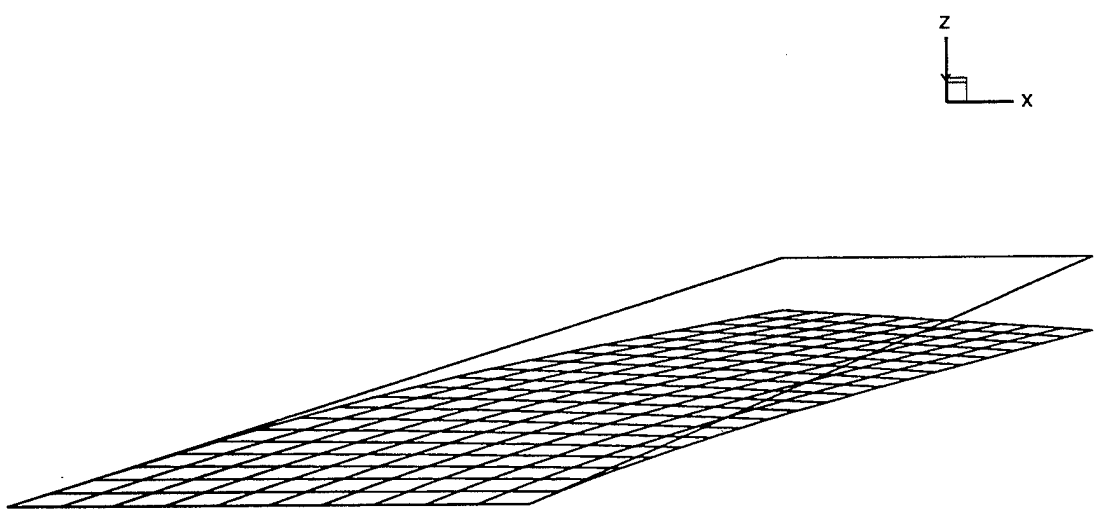


Figure 12-2. AGARD 445 Wing Aerodynamic Grid (Undeflected)

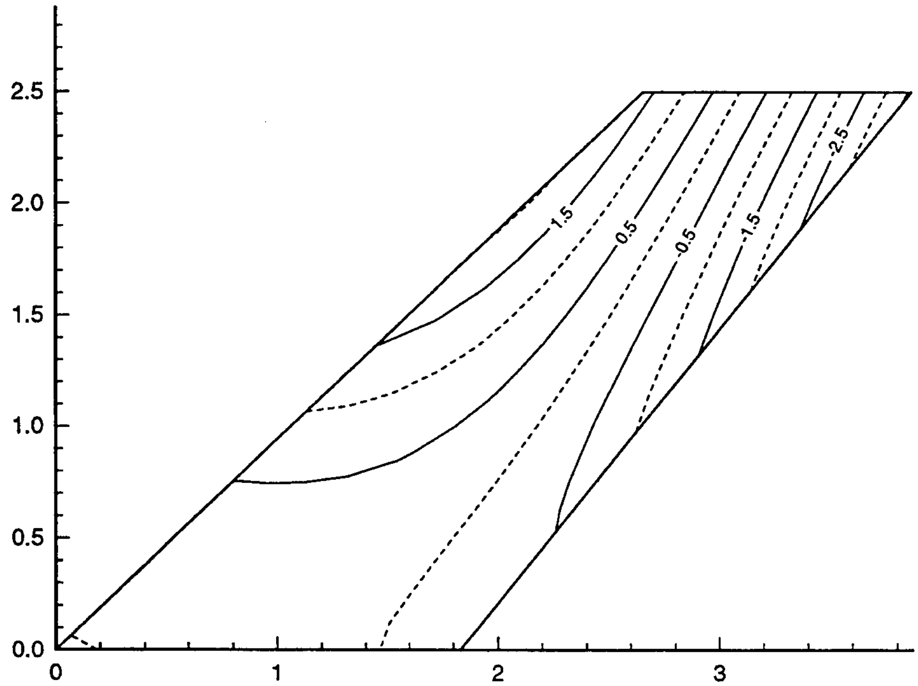


a) Structural Grid (Original) Mode Shape 1 Contours

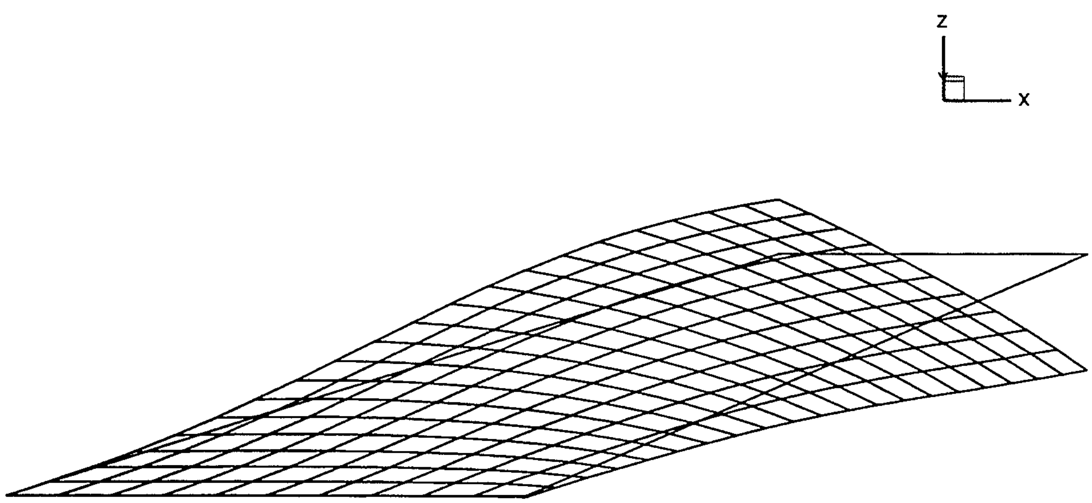


b) Structural Grid (Original) Mode Shape 1 Deflection

Figure 12-3. AGARD 445 Wing Structural Grid (Original) Mode Shape 1

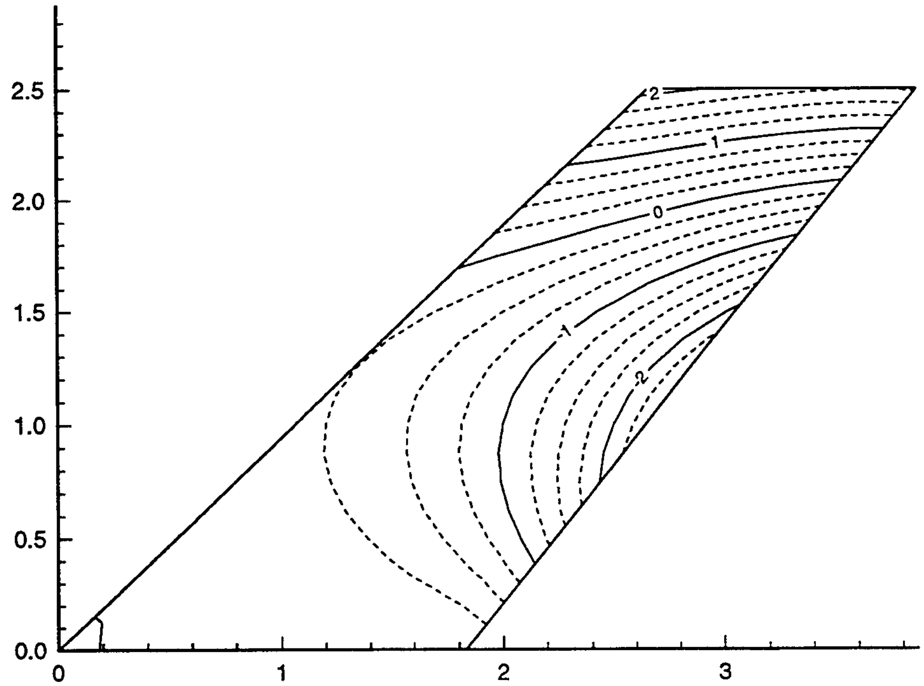


a) Structural Grid (Original) Mode Shape 2 Contours

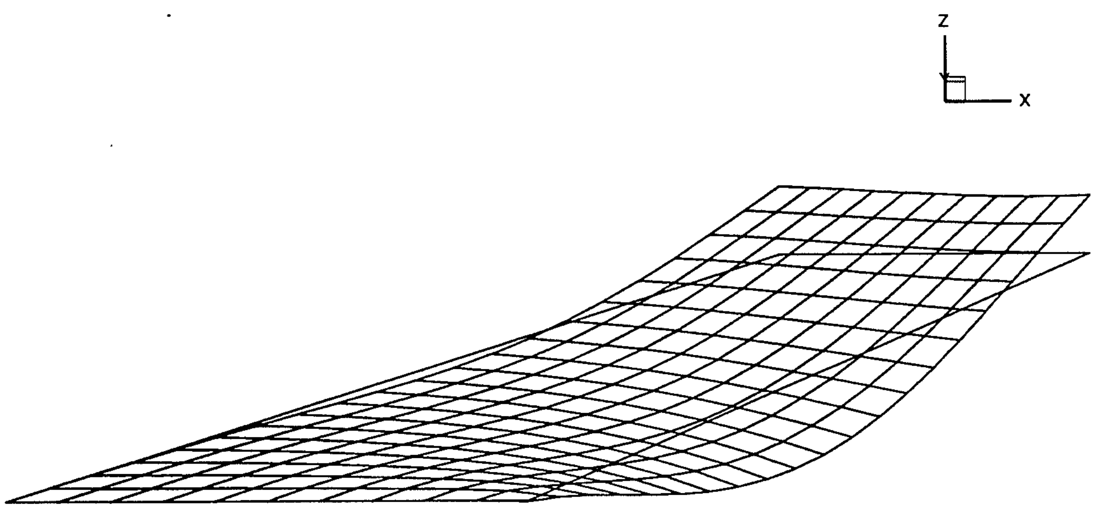


b) Structural Grid (Original) Mode Shape 2 Deflection

Figure 12-4. AGARD 445 Wing Structural Grid (Original) Mode Shape 2

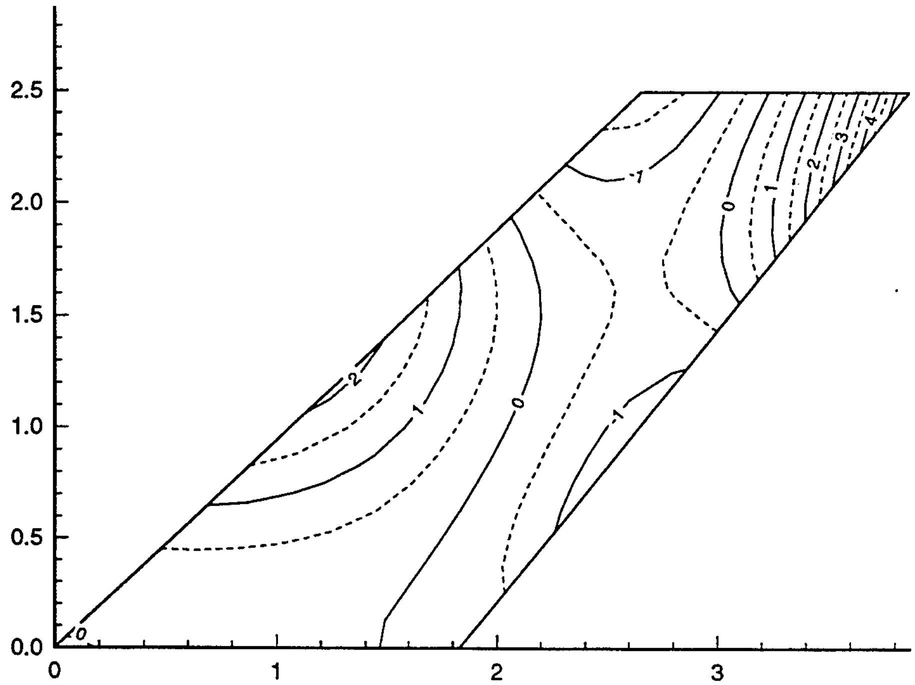


a) Structural Grid (Original) Mode Shape 3 Contours

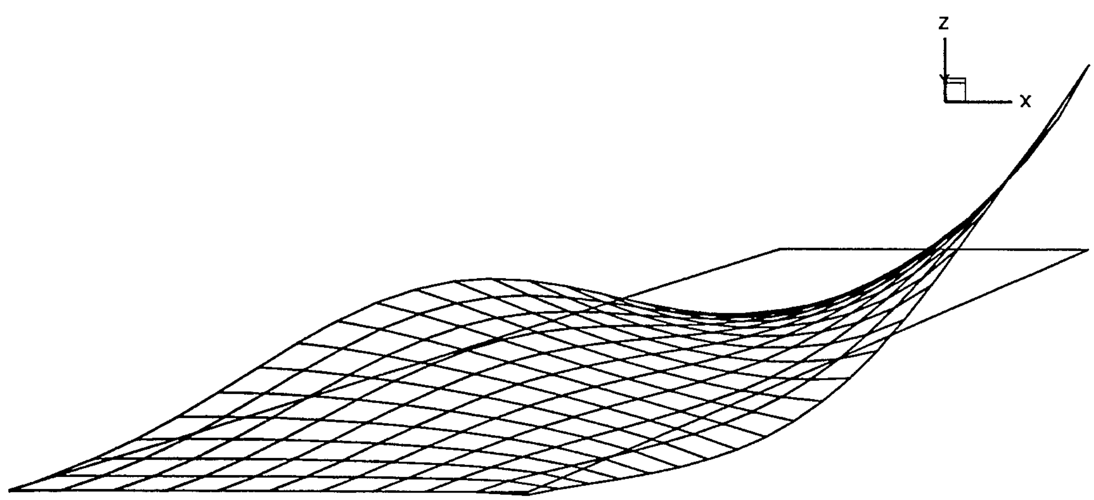


b) Structural Grid (Original) Mode Shape 3 Deflection

Figure 12-5. AGARD 445 Wing Structural Grid (Original) Mode Shape 3

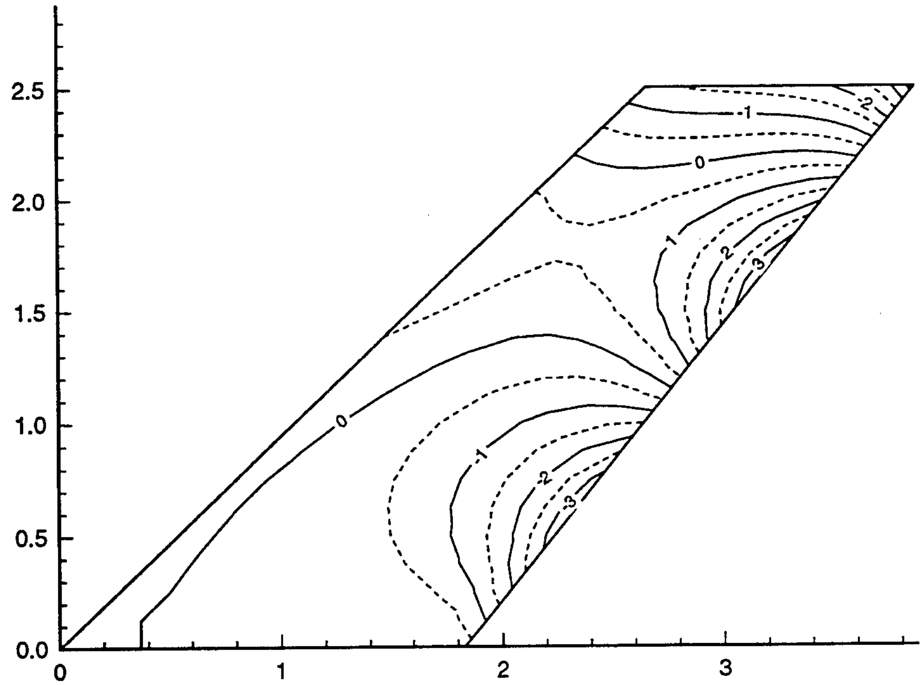


a) Structural Grid (Original) Mode Shape 4 Contours

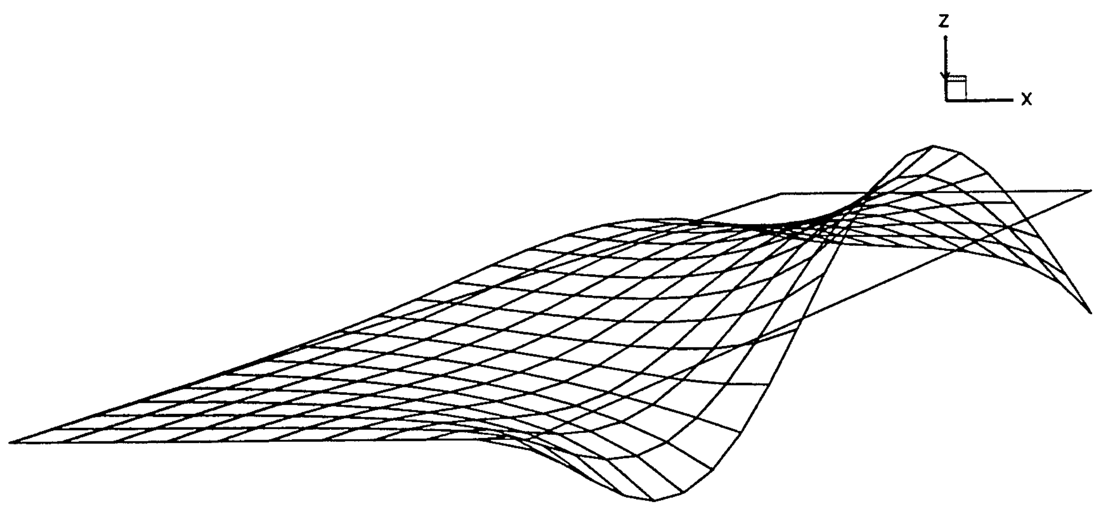


b) Structural Grid (Original) Mode Shape 4 Deflection

Figure 12-6. AGARD 445 Wing Structural Grid (Original) Mode Shape 4



a) Structural Grid (Original) Mode Shape 5 Contours



b) Structural Grid (Original) Mode Shape 5 Deflection

Figure 12-7. AGARD 445 Wing Structural Grid (Original) Mode Shape 5

12.2 Engine Liner

The engine liner test case is an application whose importance is growing as the problem of fatigue and aging aircraft becomes increasingly an issue. This particular case is based upon an experimental effort [17] to determine the cause of flutter in an engine liner. The engine liner is surrounded by a hostile aerodynamic environment. The inner flow of the liner is usually very fast (near transonic speeds) and is comprised of the high-temperature exhaust core flow from the engine. The outer side of the liner usually has ambient or bleed air flowing over it at different Mach and Reynolds numbers than the core flow. In addition, the liner dampers can trigger vortices. (Note: the liner dampers were NOT modeled during either the experimental or computational analyses.)

A structural model was developed for the liner based upon a series of small, interconnecting panels. The structural grid is shown in Figure 12-8. It was comprised of 97 nodes in the circumferential direction and 21 nodes in the streamwise direction.

The CFD surface grid, shown in Figure 12-9, was algebraically computed. This surface has 75 nodes in the circumferential direction and 45 nodes in the streamwise direction, clustered at the leading and trailing edges of the panel. As in the AGARD 445 wing test case, this test case involves pure interpolation, but includes all three Cartesian coordinate directions of motion.

The first five mode shapes were deemed to be the primary influence in the motion of the liner. These mode shapes are depicted in Figures 12-10 through 12-14. Notice that the "leading edge" of the liner is clamped (no deflections), while the "trailing edge" of the liner is permitted to move freely. The view for these figures is from the “trailing edge” of the liner looking forward. The liner forms a slight cone with the smaller radius located at the trailing edge. The use of contours is not appropriate due to the nature of the configuration. The asymmetry of the structure, superimposed with the mode shapes, makes it impossible to view contours along the entire liner surface.

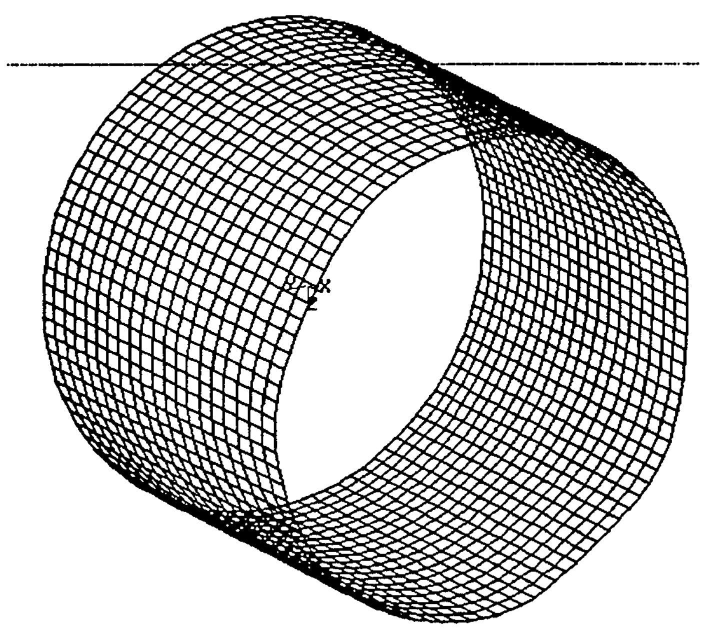


Figure 12-8. Structural Grid for the Engine Liner Applications Test Case.

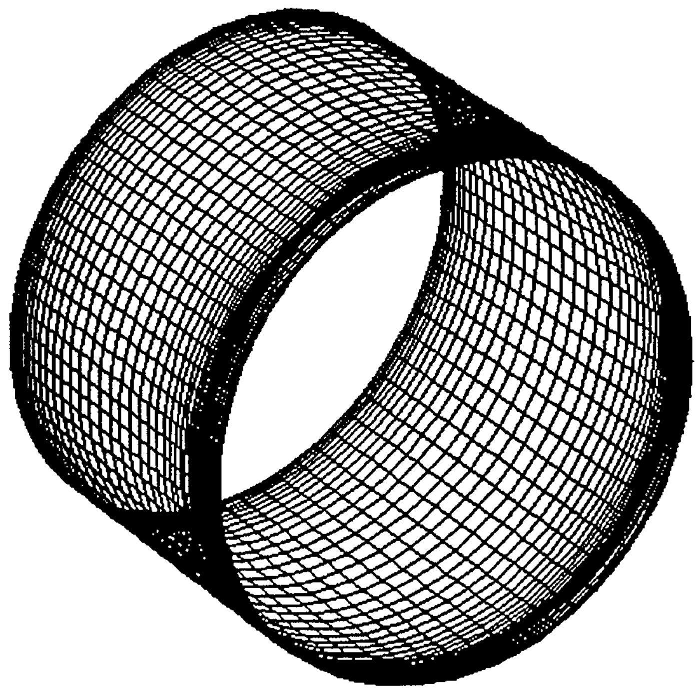


Figure 12-9. The Aerodynamic Grid for the Engine Liner Applications Test Case

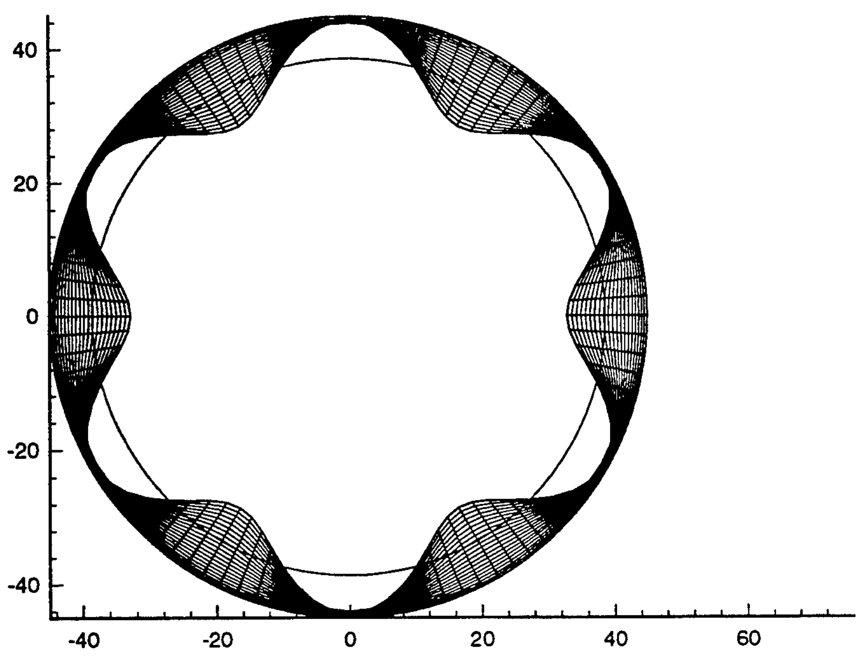


Figure 12-10. Engine Liner Mode Shape 1 for the Structural Grid

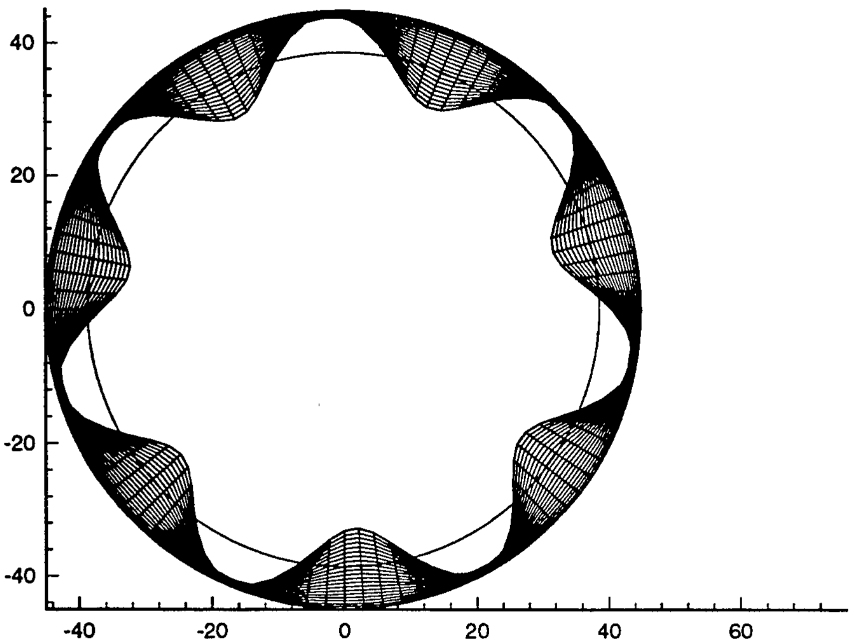


Figure 12-11. Engine Liner Mode Shape 2 for the Structural Grid

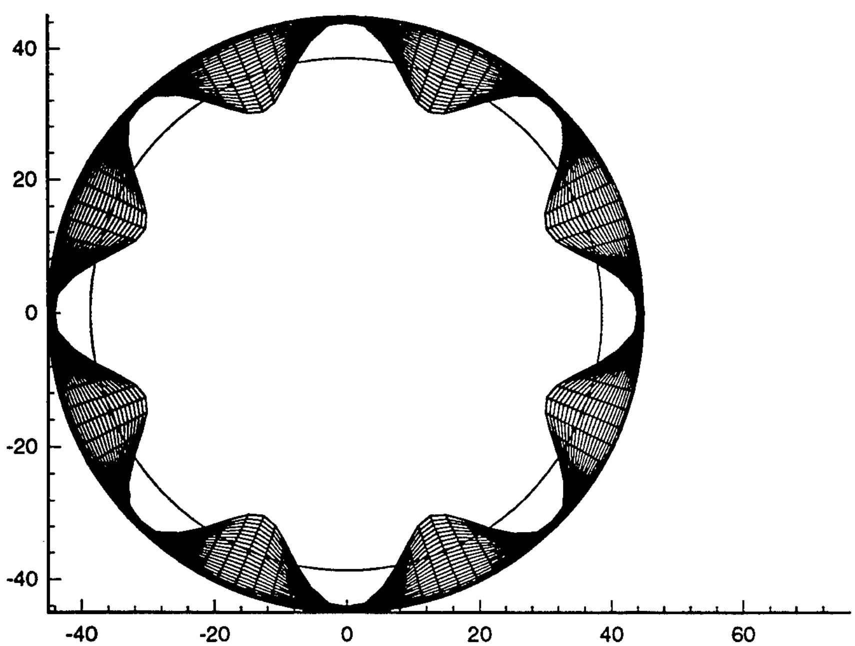


Figure 12-12. Engine Liner Mode Shape 3 for the Structural Grid

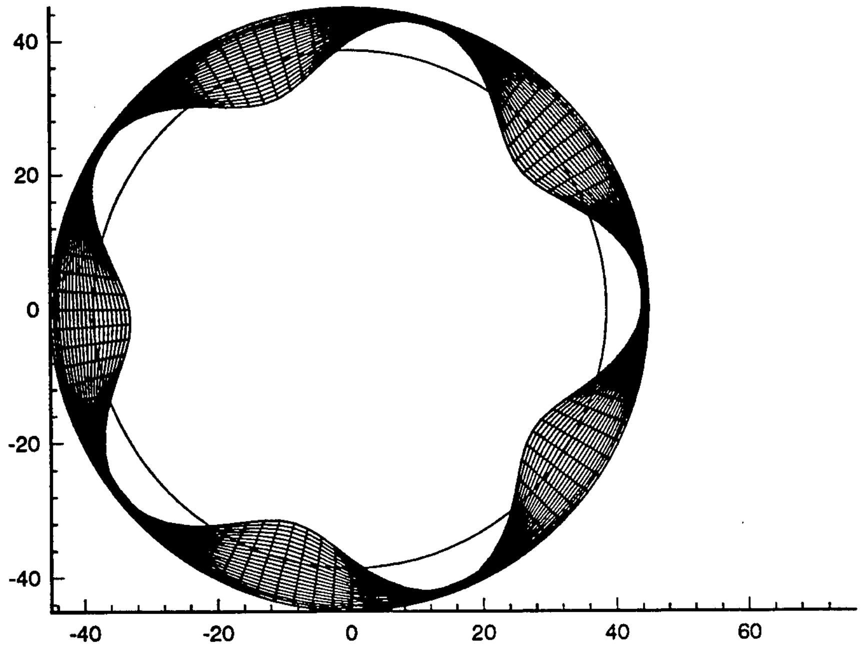


Figure 12-13. Engine Liner Mode Shape 4 for the Structural Grid

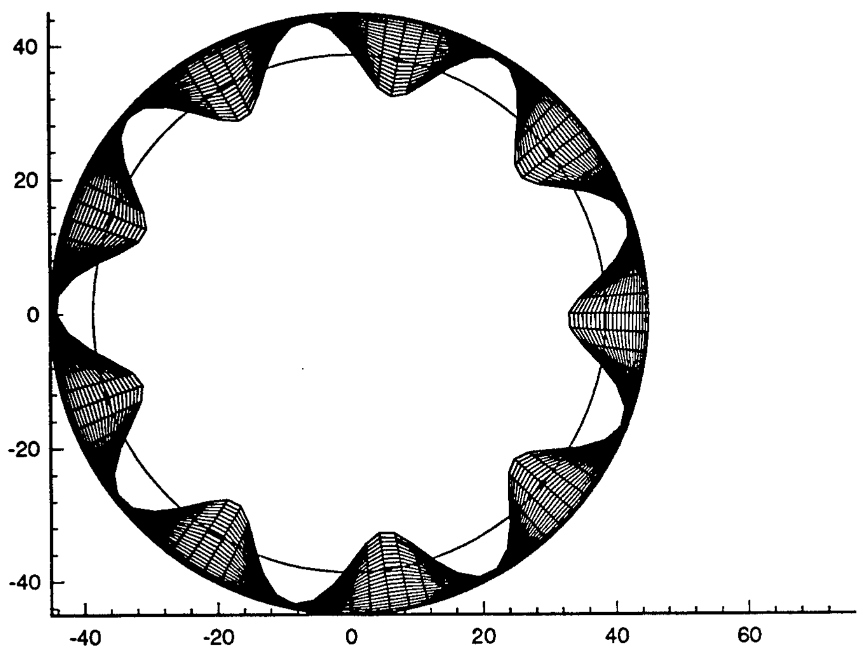


Figure 12-14. Engine Liner Mode Shape 5 for the Structural Grid

12.3 Lifting-Body Configuration

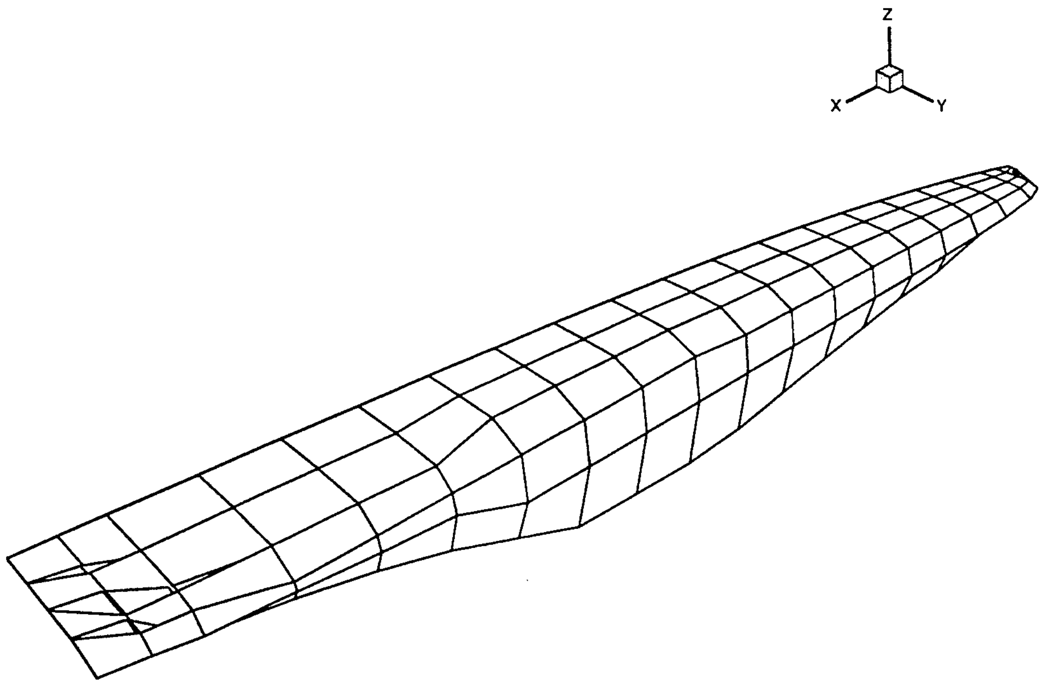
Another type of applications test case is the lifting-body. This type of vehicle is becoming ever more popular as new air/space vehicles are developed. Examples of these types of vehicles are the space shuttle, intercontinental ballistic missiles and hypersonic vehicles.

A generic lifting-body with wings was examined. This vehicle consists of separate upper and lower wing components, as well as a fuselage which is separated along the wing waterline, all of which are flexible and modeled with shell elements. This type of configuration is typical of H-H grids used in many current CFD methodologies. This configuration provided an opportunity to observe how well the methods performed on partial surfaces where matching data is critical. (For example, the leading and trailing edges of the wing should match identically.) The different components of the vehicle are shown in Figures 12-15 and 12-16 for the structural mesh and the CFD grid, respectively. The two meshes compare as follows:

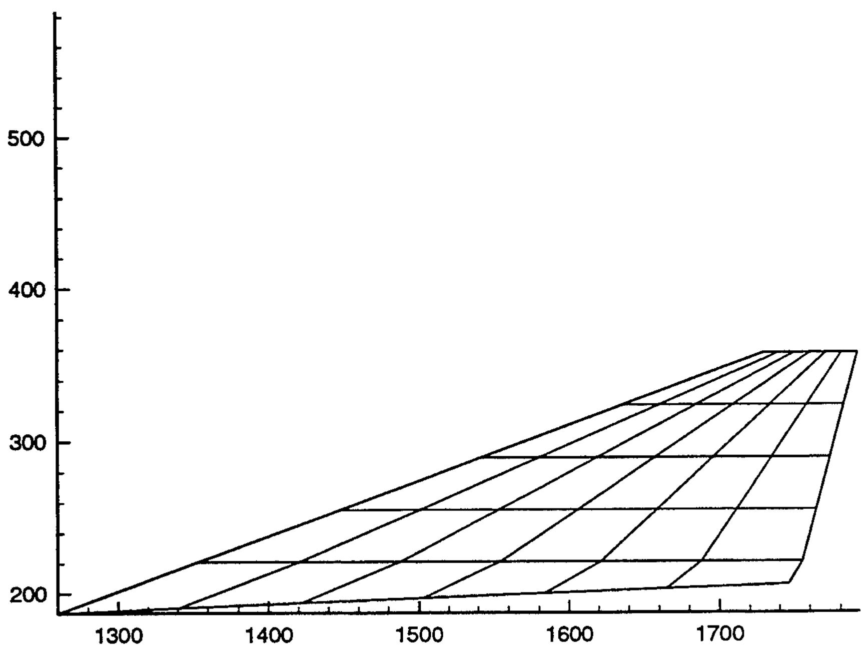
	Structural ($i \times j$)	CFD ($i \times j$)
full fuselage	9×21	109×51
split fuselage-upper		109×23
split fuselage-lower		109×29
upper wing	7×6	53×21
lower wing	7×6	53×21

where i is the number of nodes in the streamwise direction, and j is the number of nodes in the spanwise direction.

There were a total of 7 dominant modes for this model. The mode shapes which were analyzed are shown in Figures 12-17 – 12-23 for each component. The modes have been scaled to facilitate visualization.

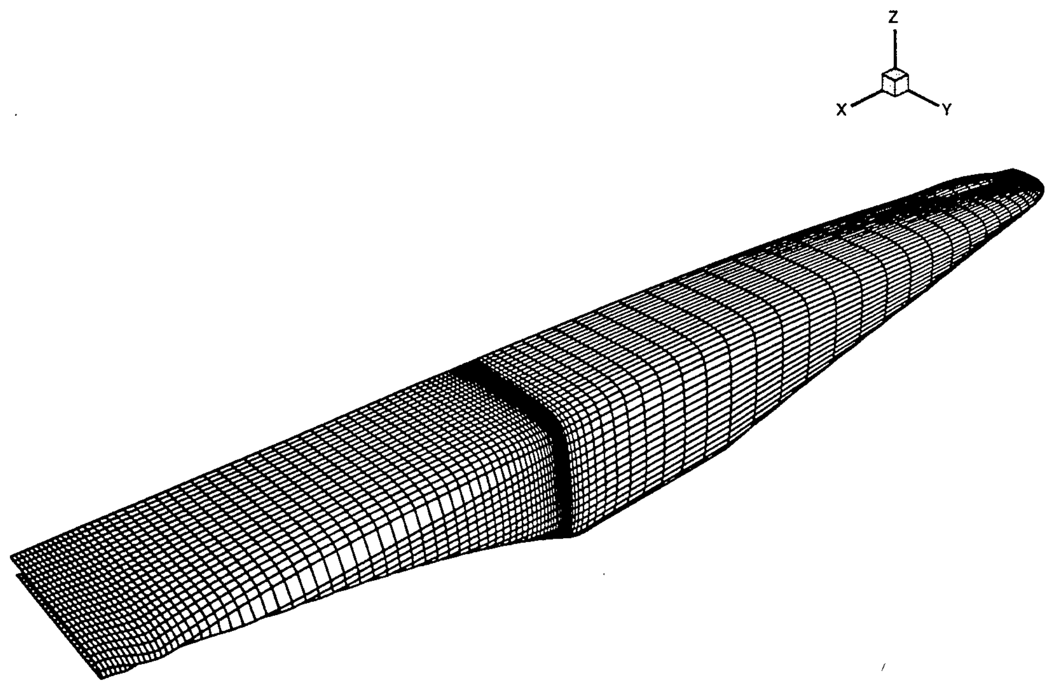


a) Fuselage Grid

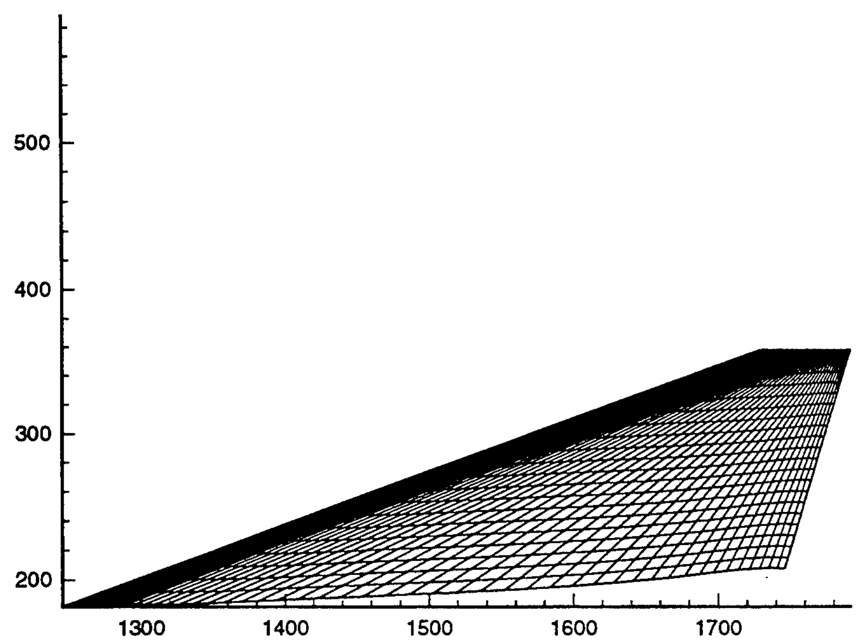


b) Wing Structural Grid

Figure 12-15. Generic Hypersonic Vehicle Structural Grid

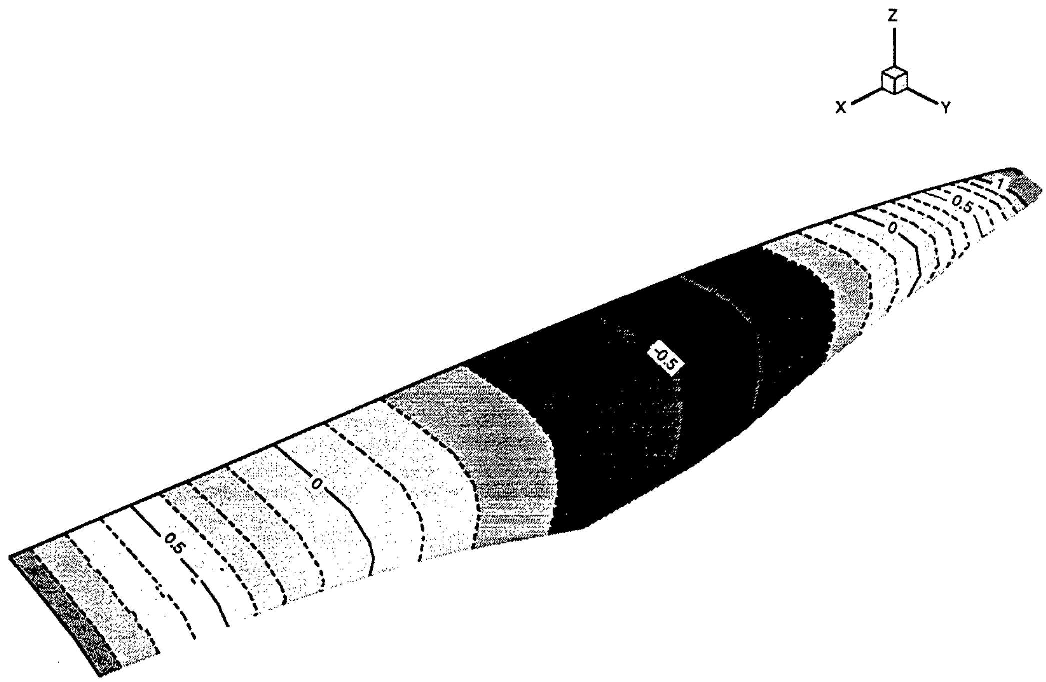


b) Fuselage Grid

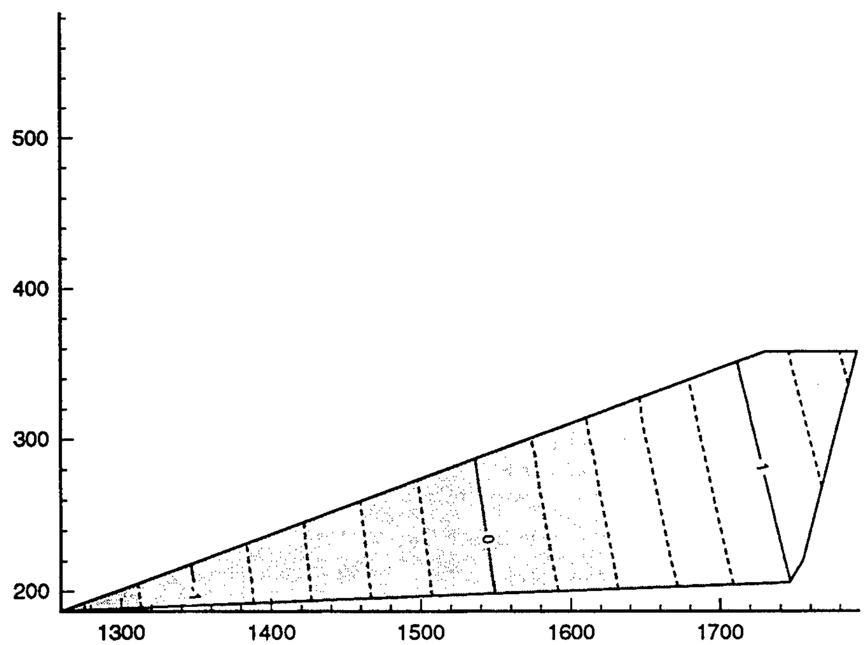


b) Wing Grid

Figure 12-16. Generic Hypersonic Vehicle Aerodynamic Grid

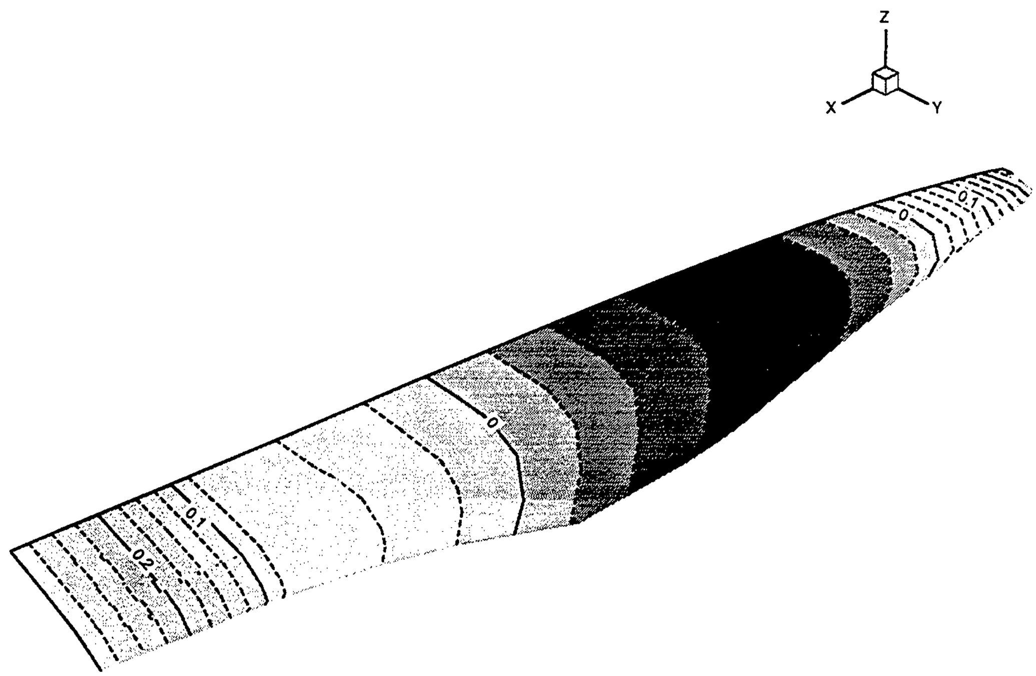


a) Fuselage

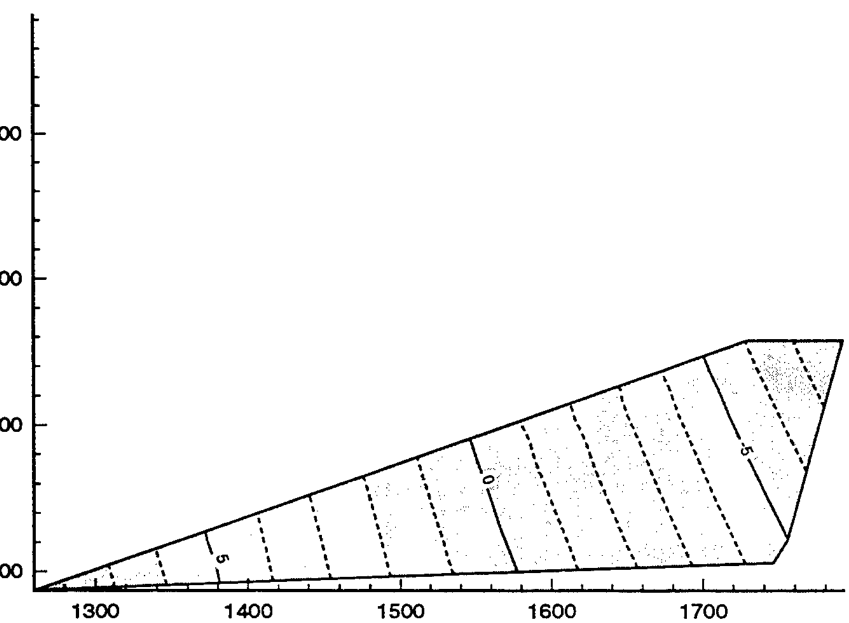


b) Wing

Figure 12-17. Generic Hypersonic Vehicle Mode 1 for the Structural Grid

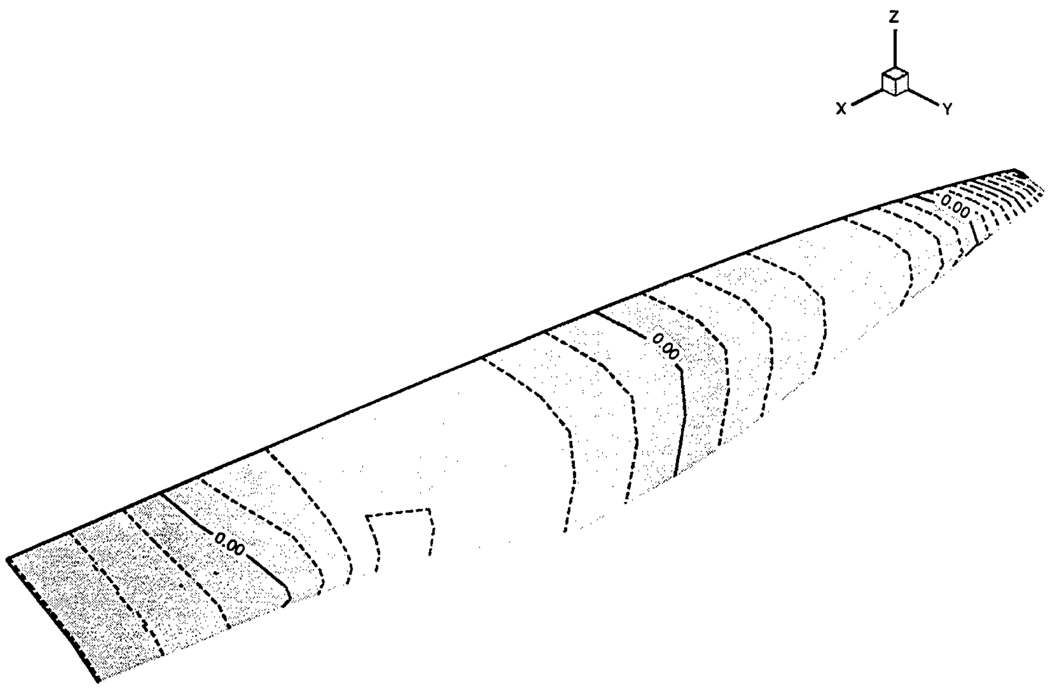


a) Fuselage

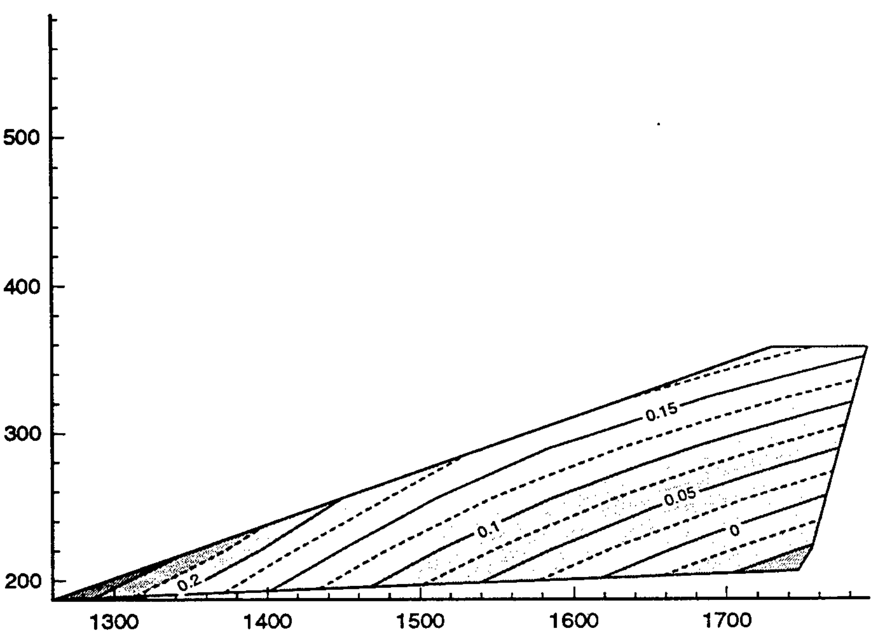


b) Wing

Figure 12-18. Generic Hypersonic Vehicle Mode 2 for the Structural Grid

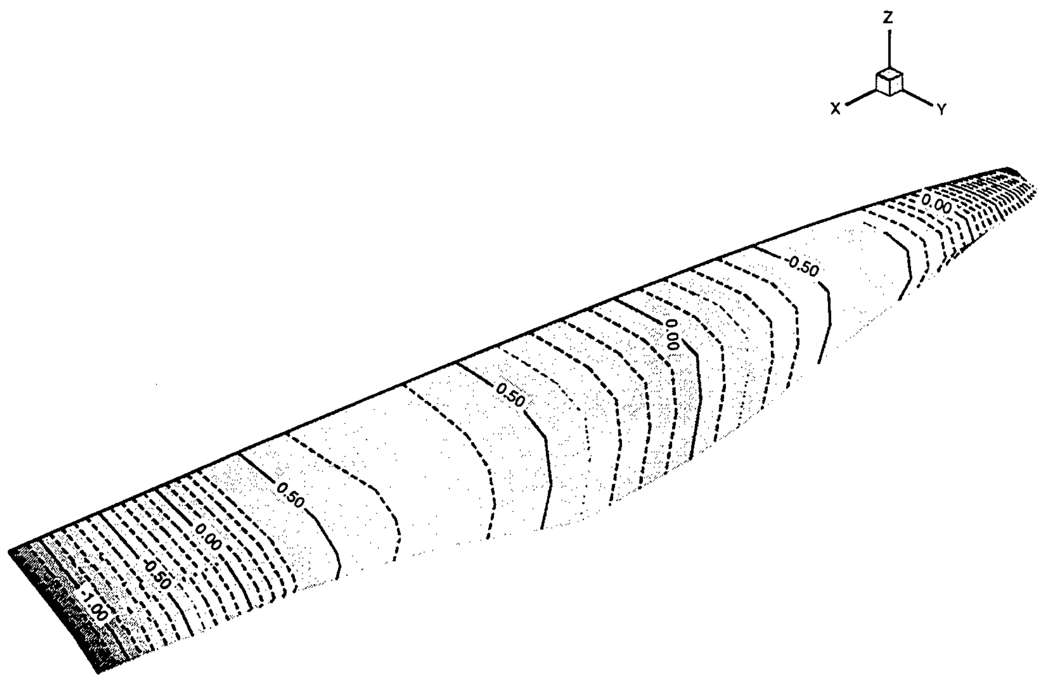


a) Fuselage

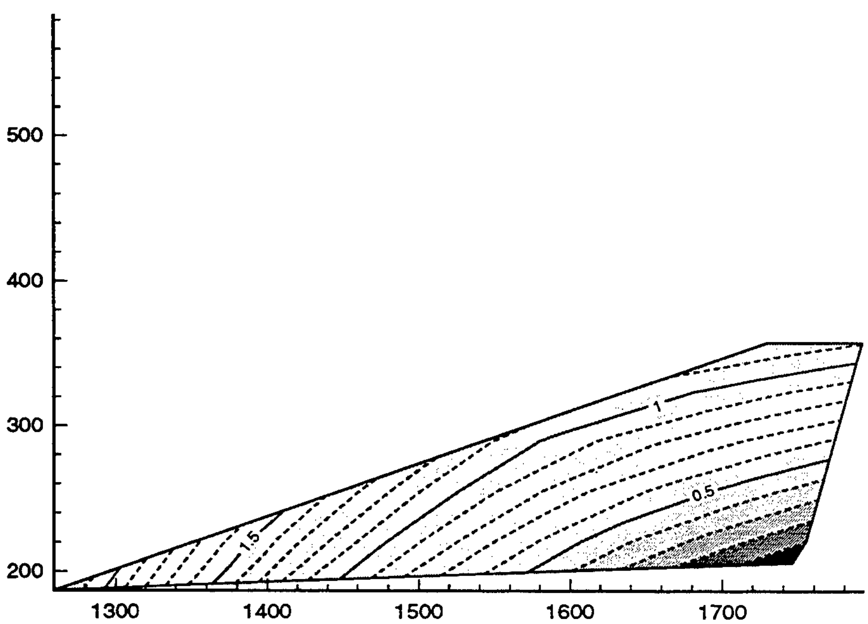


b) Wing

Figure 12-19. Generic Hypersonic Vehicle Mode 3 for the Structural Grid

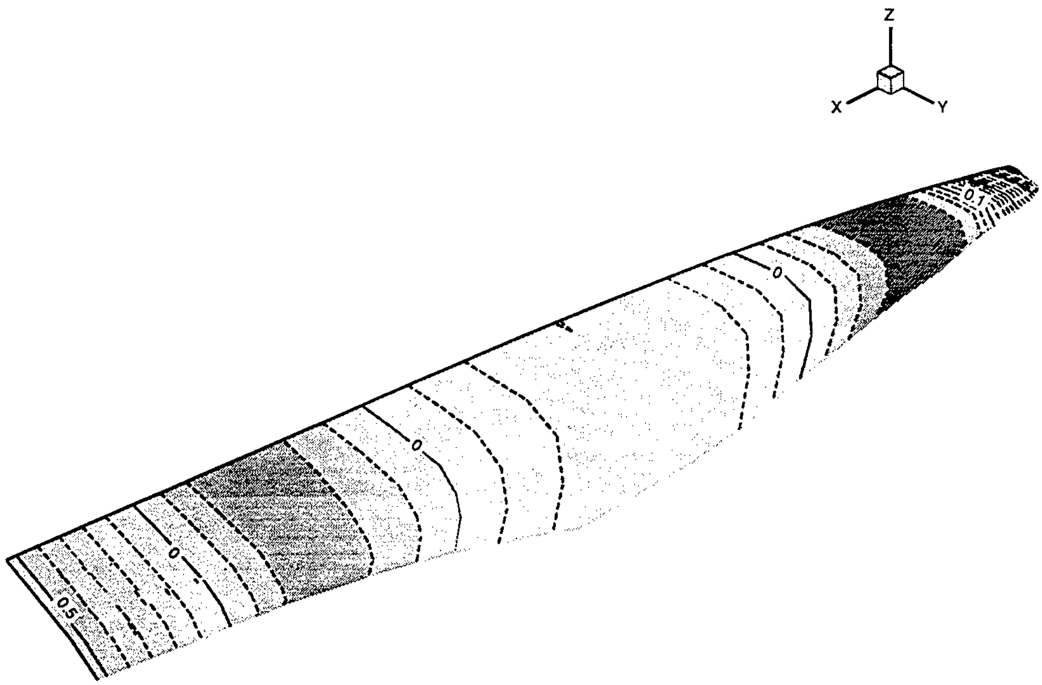


a) Fuselage

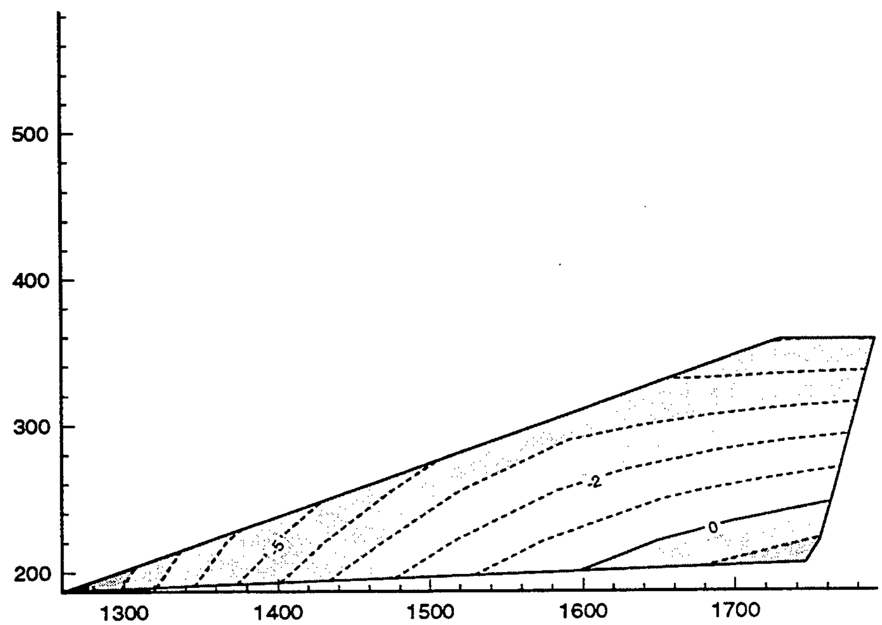


b) Wing

Figure 12-20. Generic Hypersonic Vehicle Mode 4 for the Structural Grid

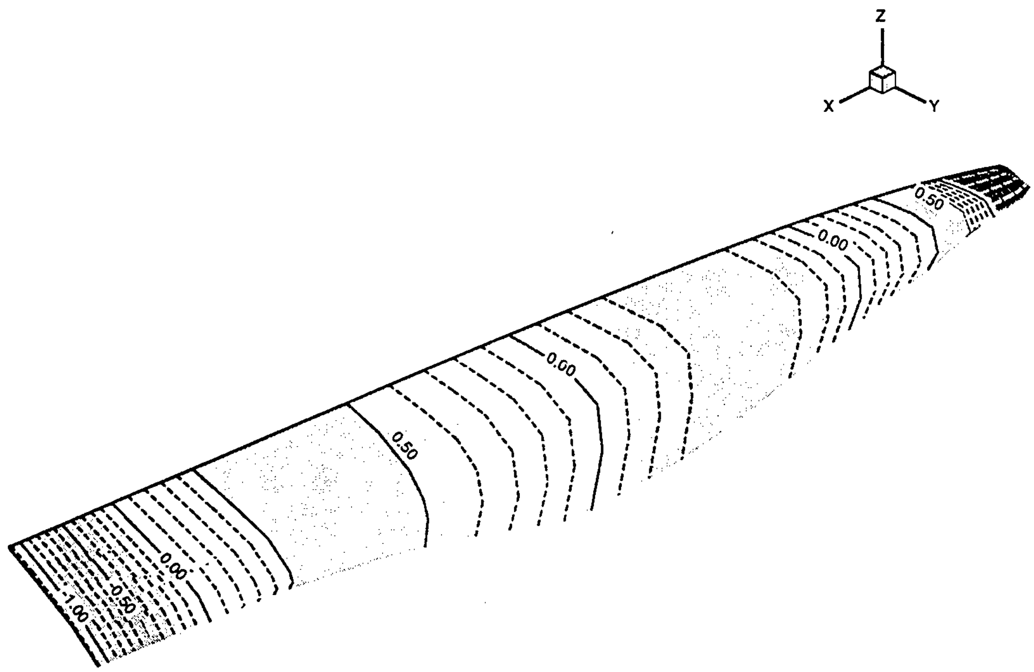


a) Fuselage

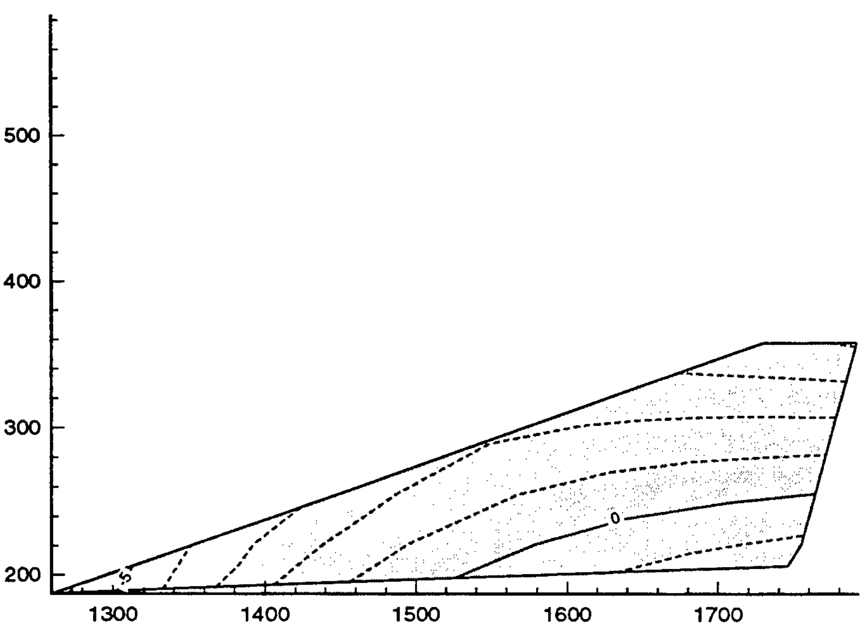


b) Wing

Figure 12-21. Generic Hypersonic Vehicle Mode 5 for the Structural Grid

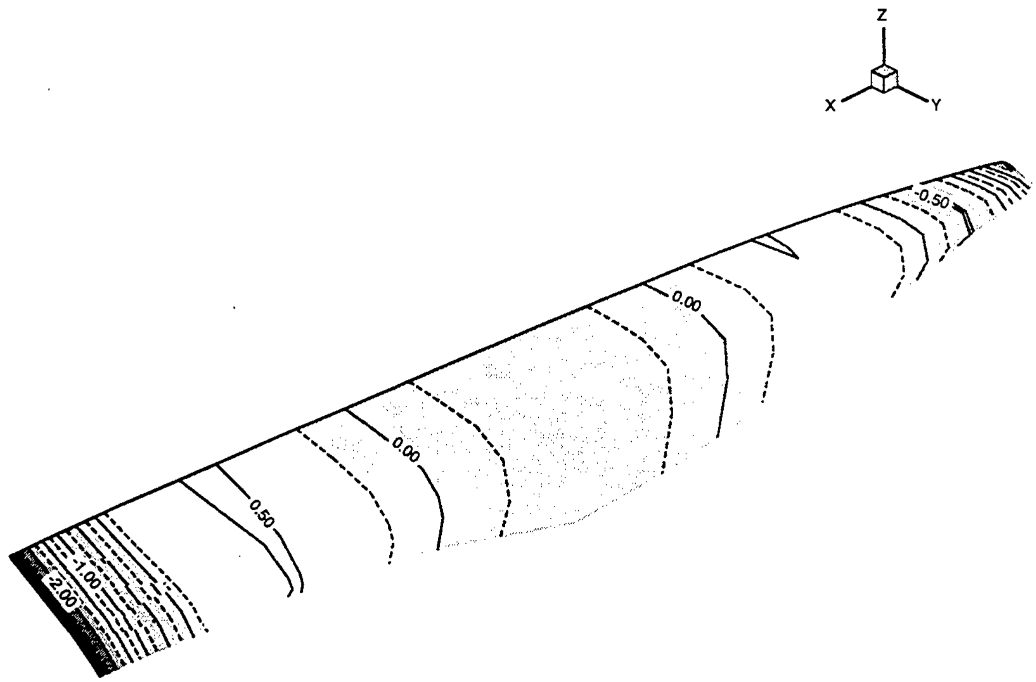


a) Fuselage



b) Wing

Figure 12-22. Generic Hypersonic Vehicle Mode 6 for the Structural Grid



a) Fuselage

b) Wing

Figure 12-23. Generic Hypersonic Vehicle Mode 7 for the Structural Grid

12.4 F-16 Wing and Strake

The next example extends the capabilities demonstrated by the first three test cases. This example examines the extrapolation abilities of each of the interpolation methods. The structural model is a flat plate which models an F-16 wing and is comprised of irregularly spaced nodes, as seen in Figure 12-24. The aerodynamic grid, shown by Figure 12-25, is a three-dimensional wing with the addition of a strake forward of the leading edge. Extrapolation is required for the strake region, as well as the leading and trailing edges of the wing. The strake extrapolation is a very exacting example of a fluid-structure interface problem.

The flat plate structural model for the wing has 24 nodes in the streamwise direction, and 17 nodes along the spanwise direction. The aerodynamic model has 118 nodes along the streamwise direction (60 nodes for the upper and lower surfaces each), and 42 nodes along spanwise direction. This aerodynamic wing is constructed differently than the previous examples. Here, the example is taken from an overset grid. The streamwise direction curves along the edge of the strake and around the wing leading edge and tip. Thus, the nodes are very highly clustered in the region of the strake edge, and while maintaining C^1 continuity, does change Cartesian coordinate directions dramatically.

The structural model consists of seven mode shapes to model the shape of the wing. These seven mode shapes are provide as Figures 12-26 through 12-32. For each figure the wing contours are modeled as part a), while the deflections are modeled as part b). Scaling is provided so that the results of the interpolations/extrapolations can be directly compared by overlaying the plots. For the deflections in part b), the outline of the undeflected aerodynamic wing/strake is provided to show the large extent of extrapolation which is necessary. Notice that there is a control surface modeled as part of the wing. The control surface deflections include very abrupt changes in the mode shape in both directions (streamwise and spanwise) along the wing. These changes are especially noticeable in modes 3, 4, 5 and 7.

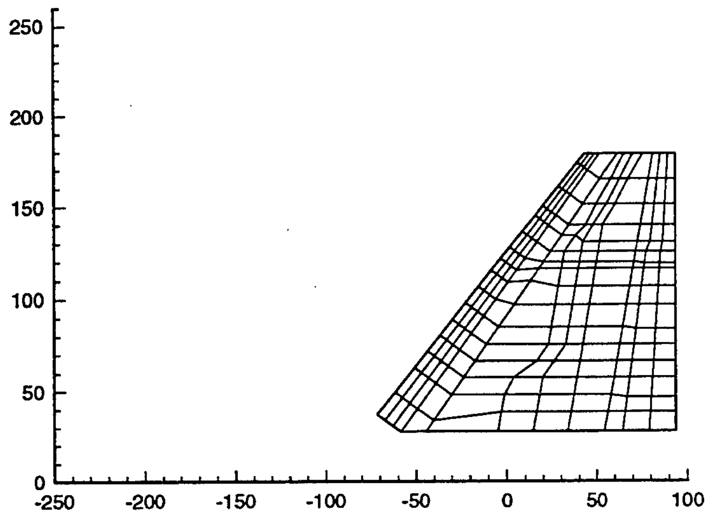


Figure 12-24. F-16 Wing Structural Grid

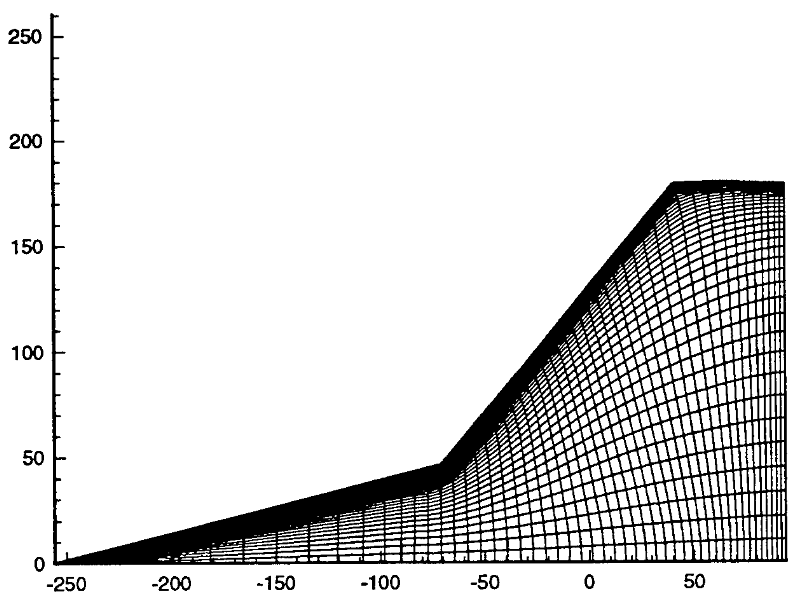
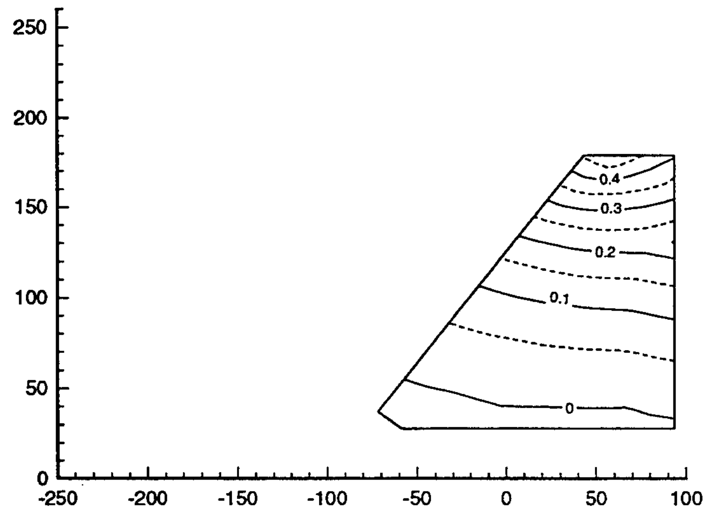
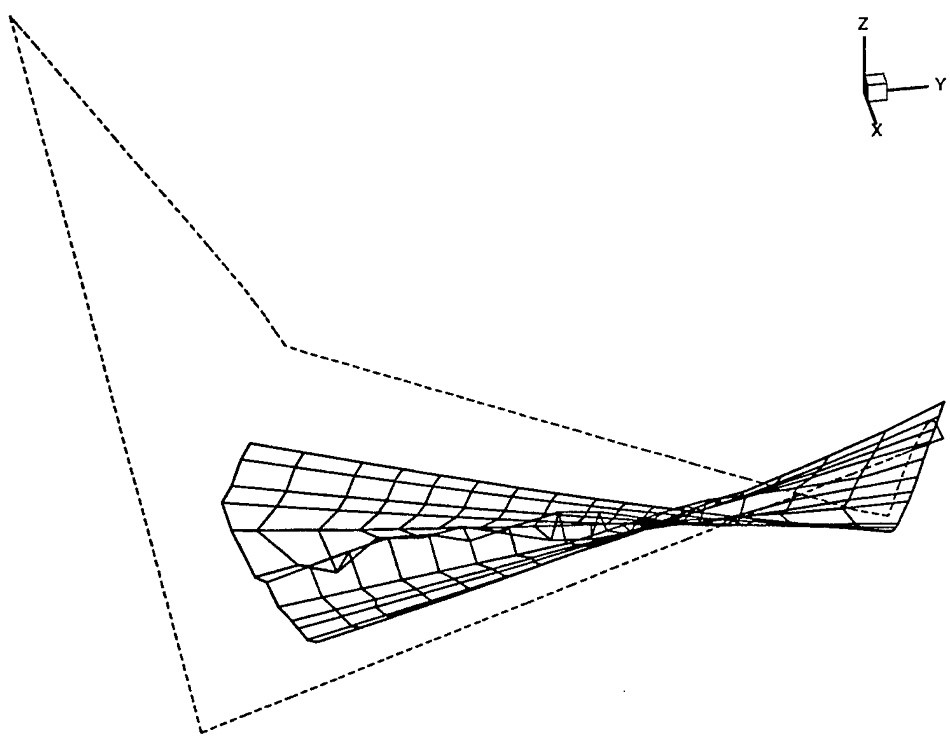


Figure 12-25. F-16 Wing/Strake Aerodynamic Grid

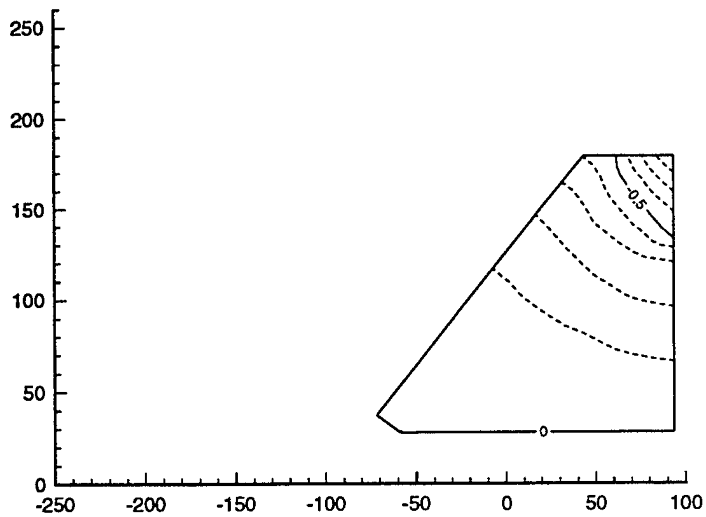


a) Mode Shape Contours

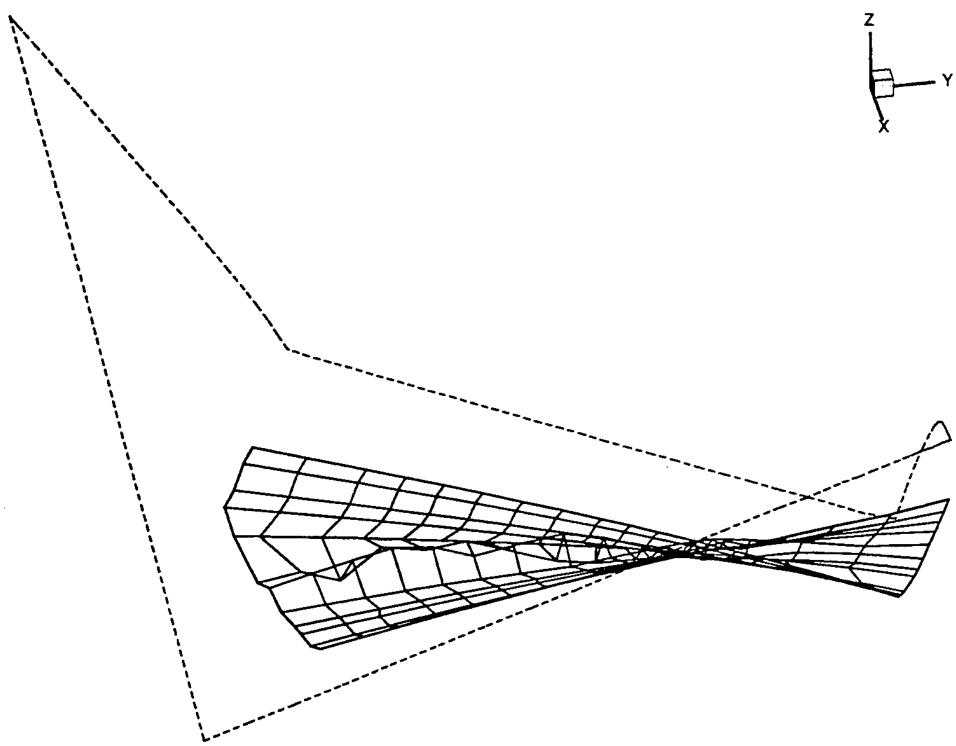


b) Mode Shape Deflections

Figure 12-26. F-16 Wing Mode Shape 1

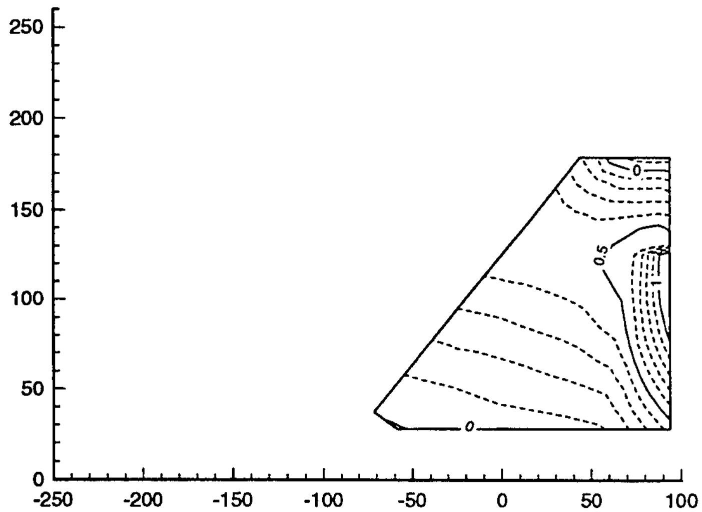


a) Mode Shape Contours

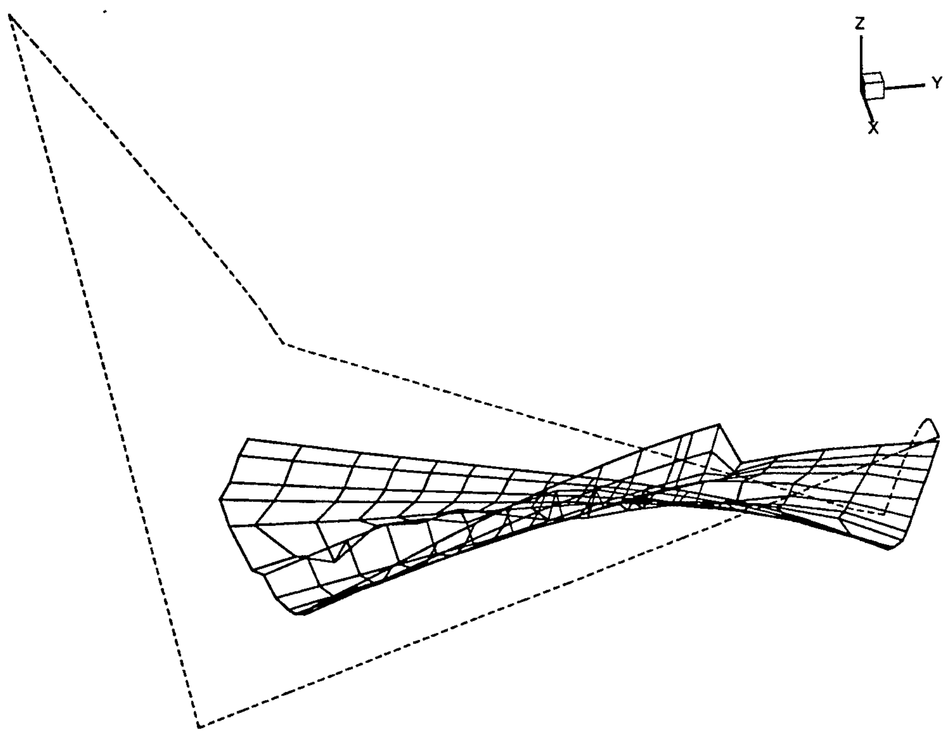


b) Mode Shape Deflections

Figure 12-27. F-16 Wing Mode Shape 2

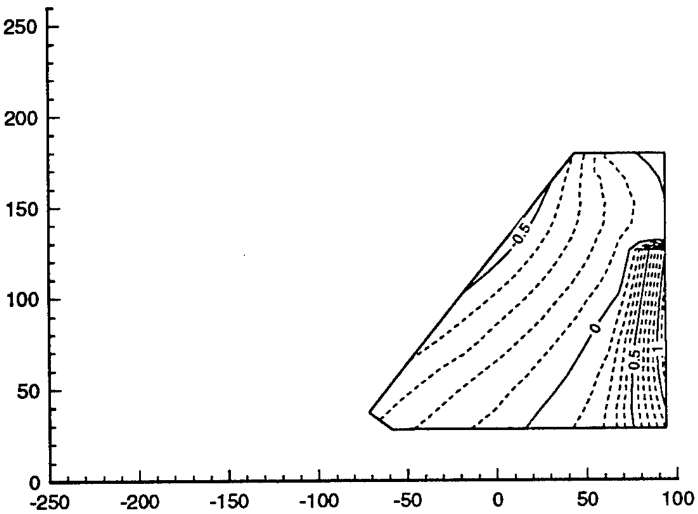


a) Mode Shape Contours

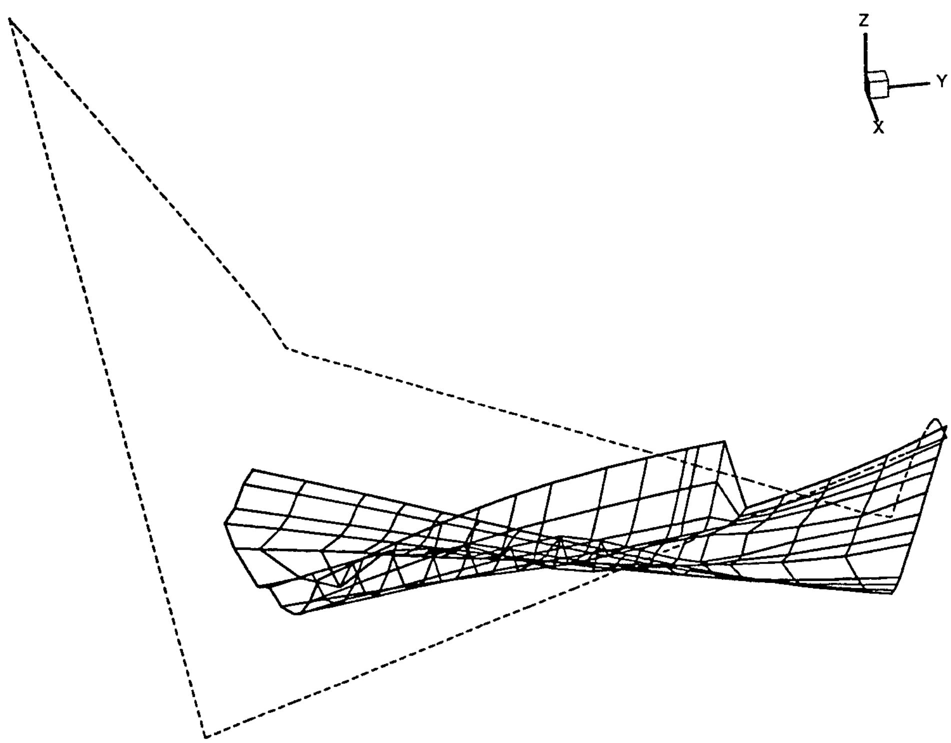


b) Mode Shape Deflections

Figure 12-28. F-16 Wing Mode Shape 3

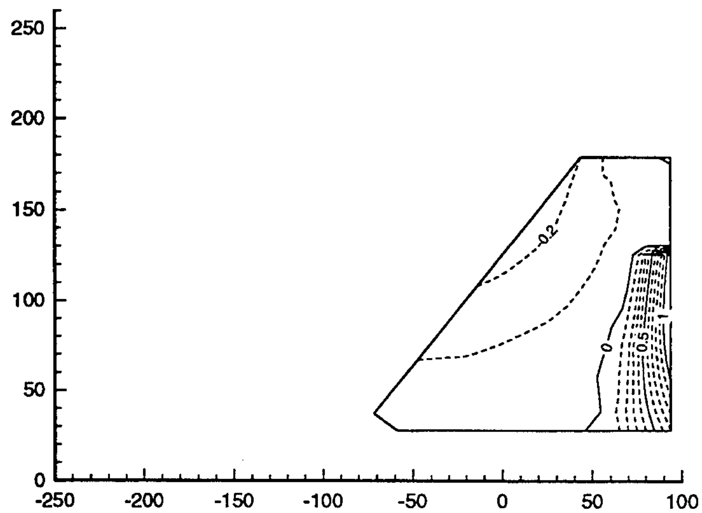


a) Mode Shape Contours

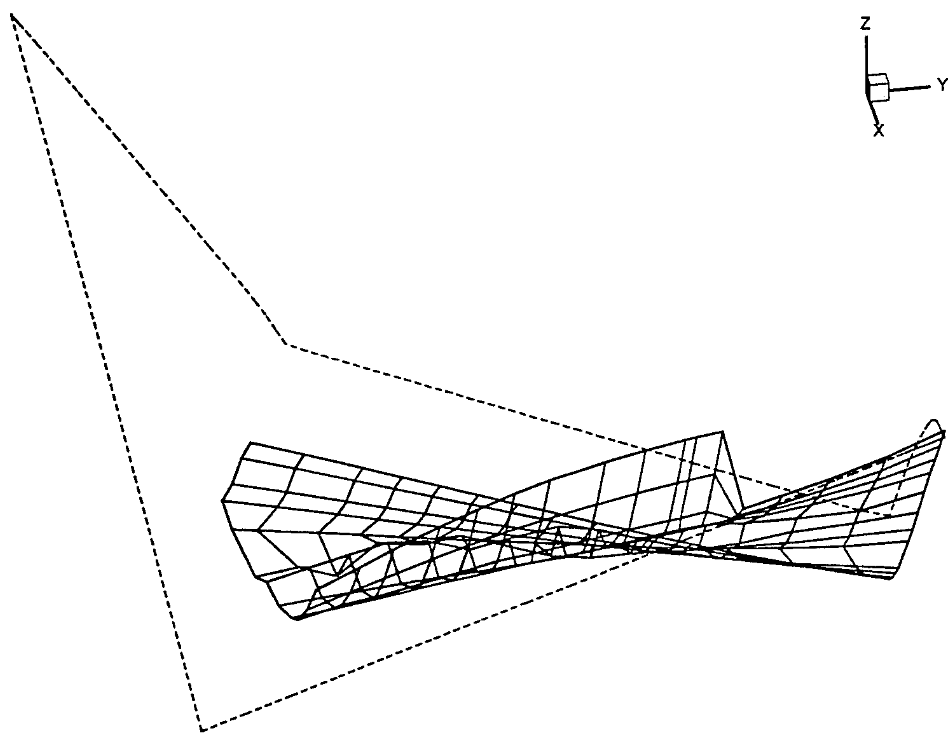


b) Mode Shape Deflections

Figure 12-29. F-16 Wing Mode Shape 4

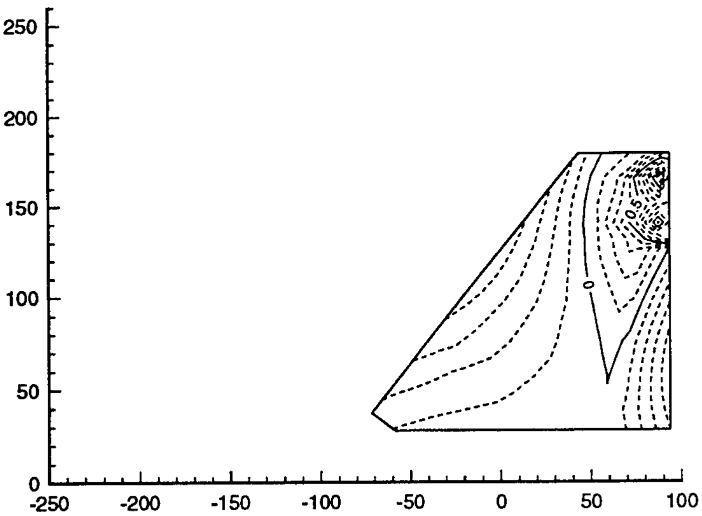


a) Mode Shape Contours

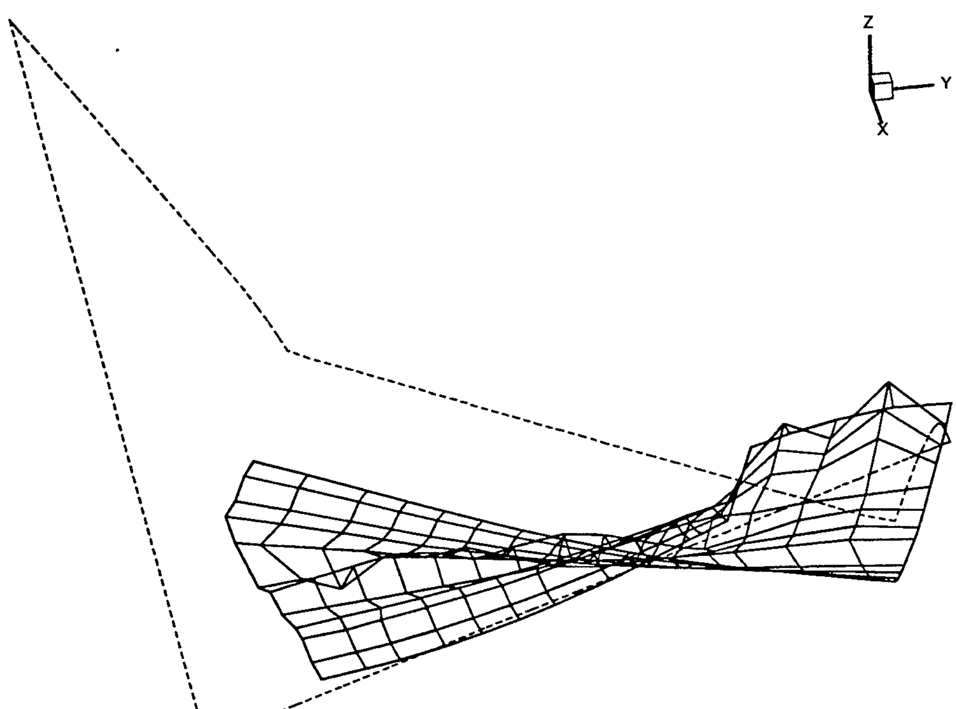


b) Mode Shape Deflections

Figure 12-30. F-16 Wing Mode Shape 5

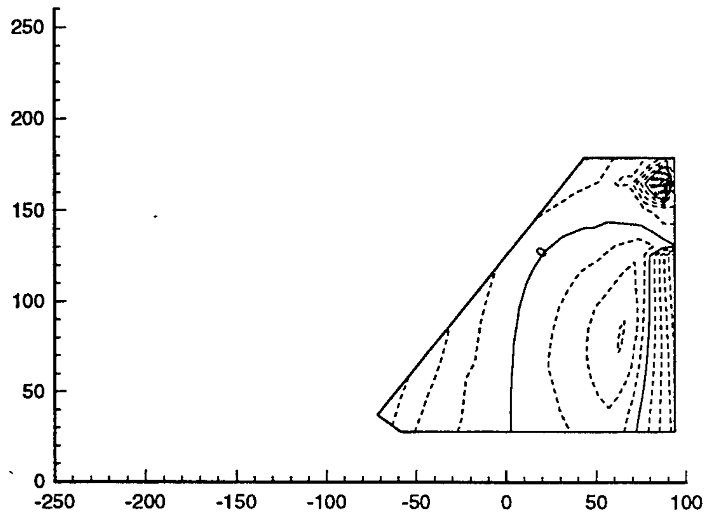


a) Mode Shape Contours

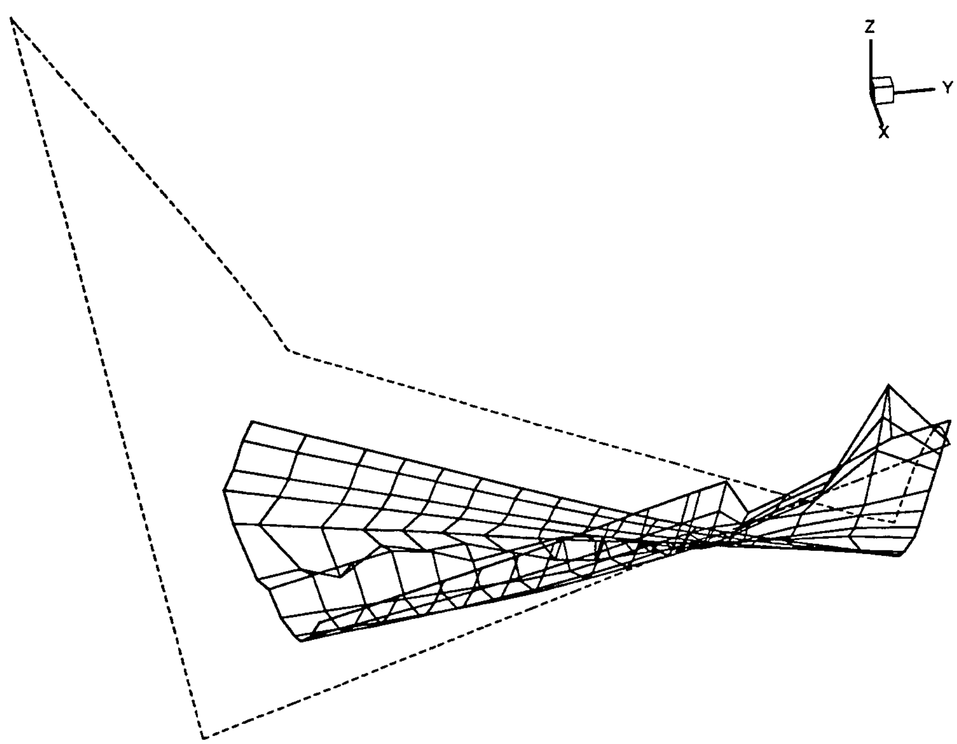


b) Mode Shape Deflections

Figure 12-31. F-16 Wing Mode Shape 6



a) Mode Shape Contours



b) Mode Shape Deflections

Figure 12-32. F-16 Wing Mode Shape 7

12.5 F-16 Flexible Wing with Rigid Body

The final applications test case involves an F-16 flexible wing attached to a rigid body. Unlike all of the previous test cases, the wing structure is predicted using influence coefficients rather than mode shapes. In addition, this applications case is also used to examine the capability of the interpolation routines to integrate loads from the CFD wing to the structural node points.

The structure of the wing is predicted by 28 influence coeffients located at 7 nodes in the spanwise direction and 4 nodes in the streamwise direction, as shown in Figure 12-33. The wing is predicted as a flat plate surface - that is the influence coefficients are taken to act normal to the direction of the undeflected structural surface.

The aerodynamic grid used in this test case is shown in Figure 12-34. The aerodynamic grid has 107 points wrapped from upper trailing edge to lower trailing edge, with 53 points on each surface. There are 25 nodes which determine the shape of the spanwise surface grid.

Once set of influence coefficients are presented in this report. They are shown as contours for the structural grid in Figure 12-35. Pressure contours from an aerodynamic run is provided for comparison with the final loads contours in Figure 12-36.

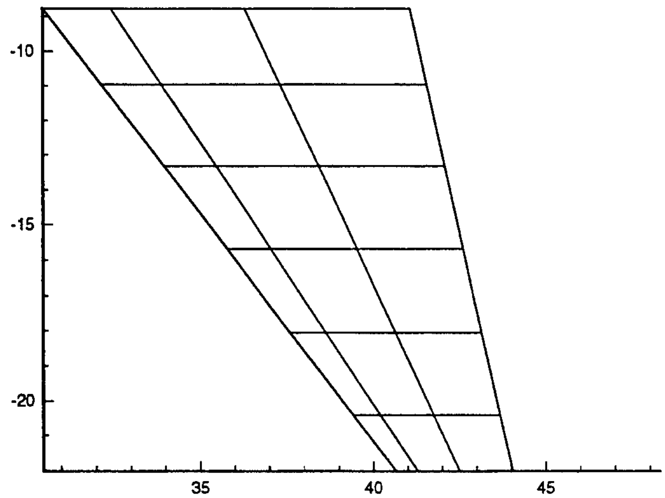


Figure 12-33. F-16 Flexible Wing Structural Mesh

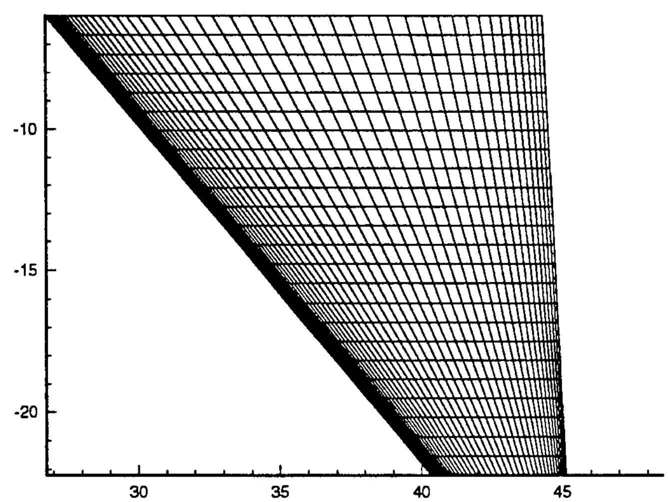


Figure 12-34. F-16 Flexible Wing Aerodynamic Surface Grid

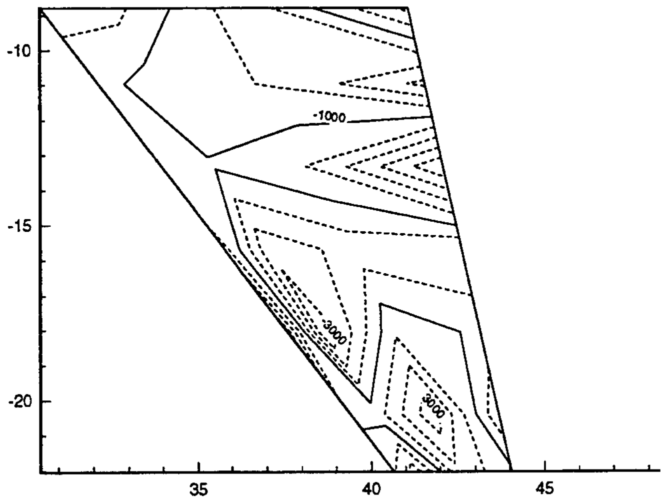


Figure 12-35. F-16 Wing Structural Influence Coefficients
(Plotted on Figure 12-33 Grid)

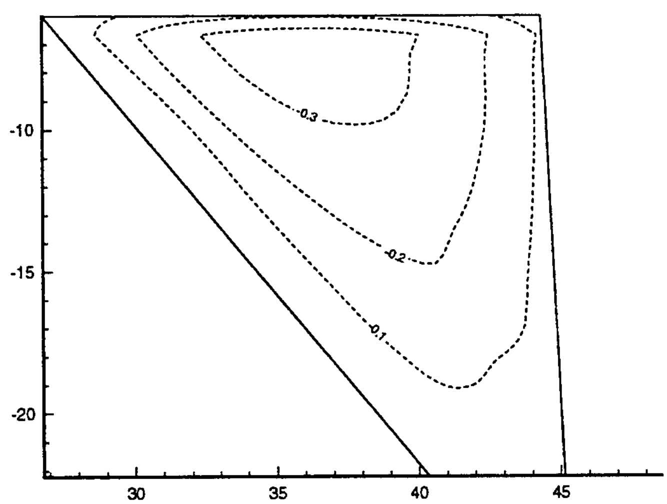


Figure 12-36. F-16 Wing Pressures
(Plotted on Figure 12-34 Grid)

13. RESULTS OF THE APPLICATIONS TEST CASES

This chapter describes the results of the applications test cases described in Chapter 12 to which the mathematical formulations were applied. It was not possible to test every algorithm discussed in Chapters 4 through 9 to each applications test case. This was due to the nature of the algorithm implementation or due to limitations on the computational workstations used in this study. The limitations are duly noted. Each section within this chapter follows the outline of the previous chapter, i.e., the AGARD 445 Wing test case is described in Section 12.1 and the results are discussed in Section 13.1, etc. All of the plots in this chapter were generated using the identical scales as the original data in Chapter 12 so that direct comparisons can be made. The plots for each test case are provided at the end of the appropriate section.

13.1 AGARD 445 Wing

The AGARD 445 wing mode shapes were interfaced using the infinite-plate spline (IPS), multiquadrics (MQ), thin-plate spline (TPS), and non-unform B-spline (NUBS) methods. The inverse isoparametric method (IIM) should have worked for this case, but the search algorithm did not work for several of the nodes. The (finite-plate spline) method was too large to run on any of the workstations applied on this study.

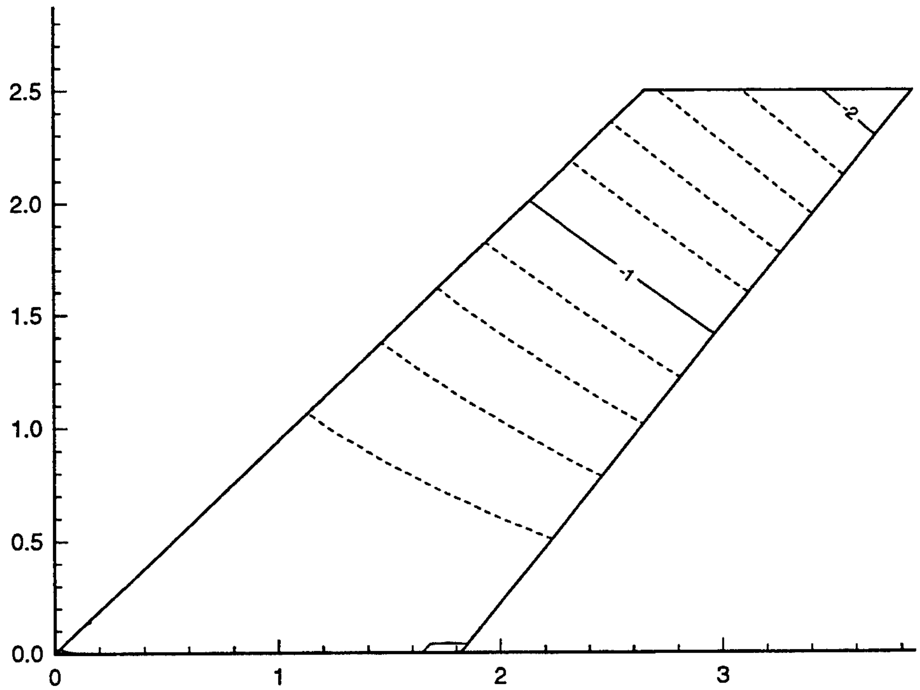
The results of the these methods are given in Figures 13-1 to 13-5 for the infinite-plate spline method, Figures 13-6 to 13-10 for the multiquadrics method, Figures 13-11 to 13-15 for the thin-plate spline method and Figures 13-16 to 13-20 for the non-uniform B-spline. The format of these plots follow the format described in Chapter 12, for Figures 12-3 to 12-8.

The primary difference in the prescribed modes and the interpolated modes is the outboard shift of the zero deflection point from the actual root line. This shift is characterized by the zero contour line at the root. It is particularly noticeable for mode shape 2. The "0" contour line extends to the wing root from the wing span. None of the interface schemes accurately reproduces this contour line. The "0" contour always ends short of the wing root.

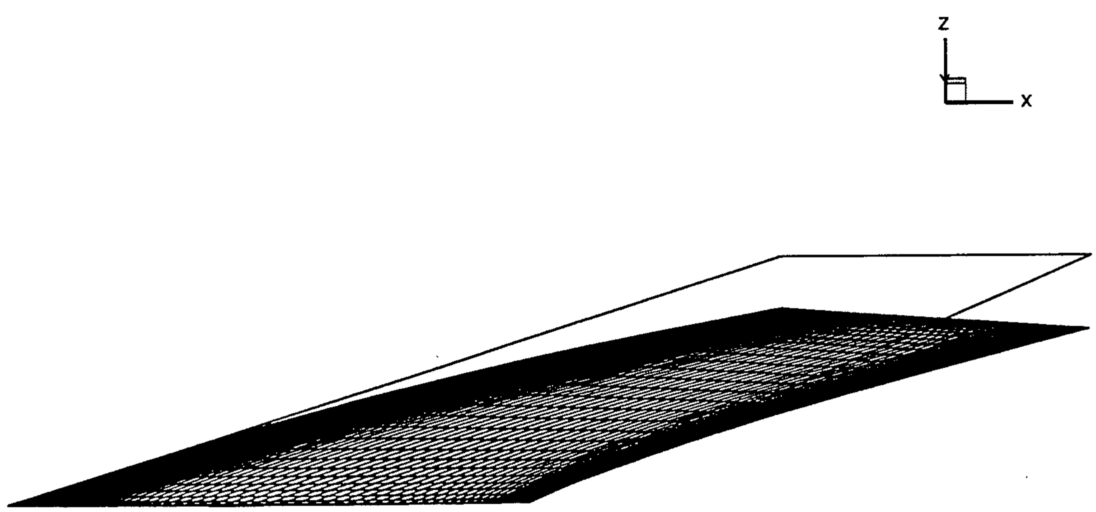
Some slight oscillations are also noticeable in the oscillations, particularly near the tip for mode shape four and near the trailing edge for mode shape 5. The infinite-plate spline method has the most obvious oscillations, while the thin-plate spline method appears to do the most accurate interpolation. This is very interesting since these two methods are based upon the same derivation. Therefore, the implementation of the scheme plays an important role in the accuracy.

Table 13.1 Maximum Deflections For the AGARD 445 Wing Mode Shapes

Mode Shape	Original	Infinite-Plate Spline	NUBS	TPS	MQ
1	2.240	2.240	2.240	2.240	2.240
2	3.597	3.597	3.597	3.597	3.597
3	2.477	2.453	2.470	2.462	2.424
4	5.774	5.774	5.774	5.774	5.774
5	3.762	3.762	3.762	3.762	3.762

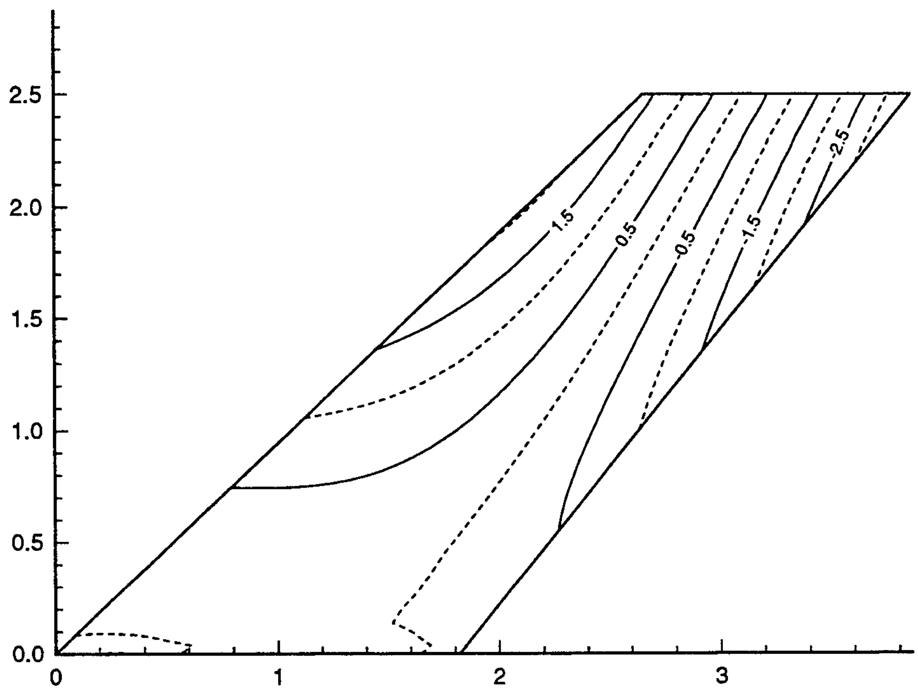


a) Mode Shape Contours

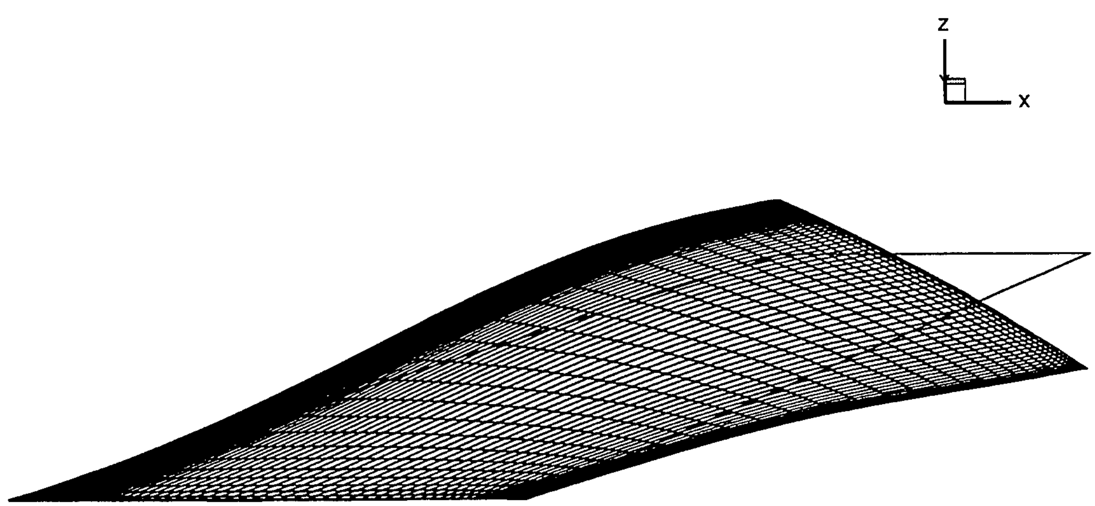


b) Mode Shape Deflection

Figure 13-1. Infinite-Plate Spline Results for AGARD 445 Wing, Mode Shape 1

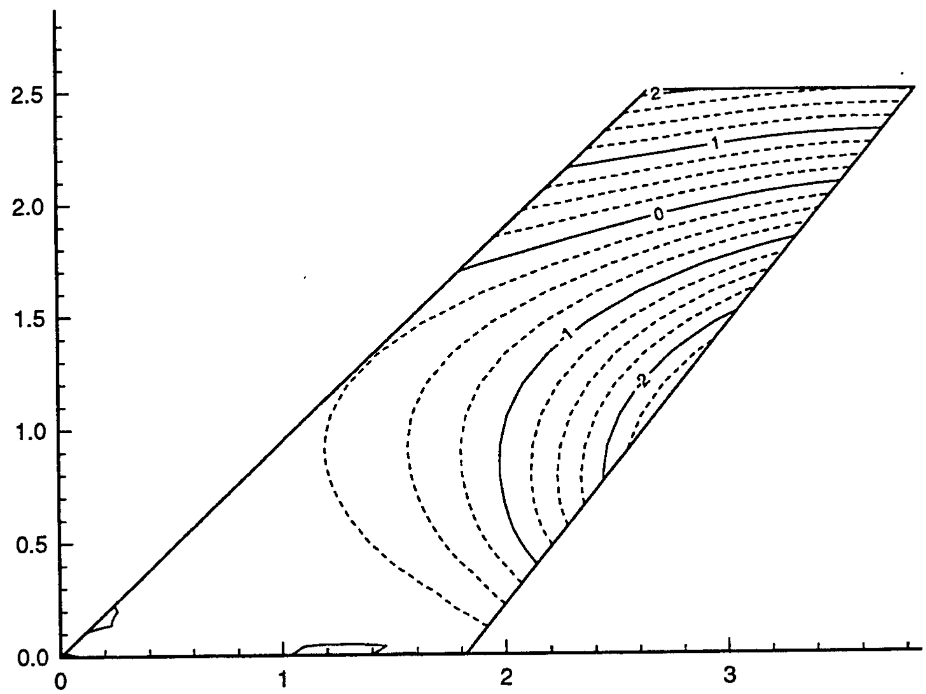


a) Mode Shape Contours

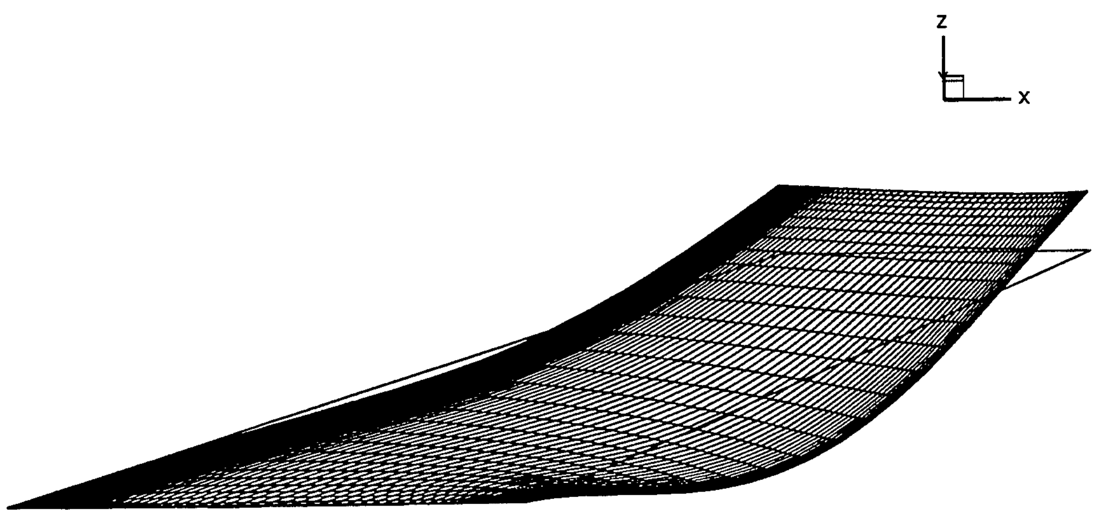


b) Mode Shape Deflection

Figure 13-2. Infinite-Plate Spline Results for AGARD 445 Wing, Mode Shape 2

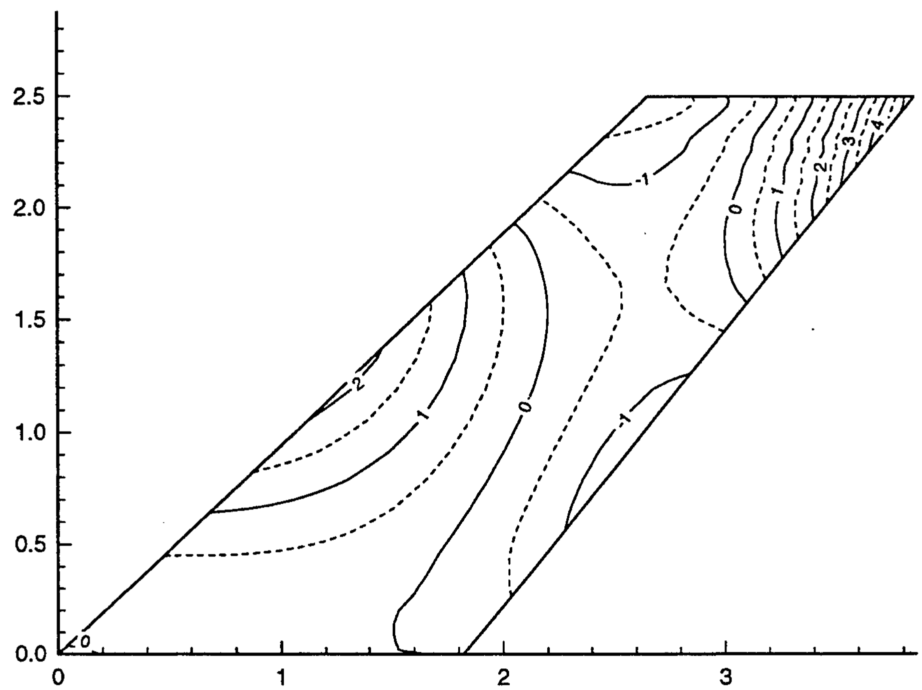


a) Mode Shape Contours

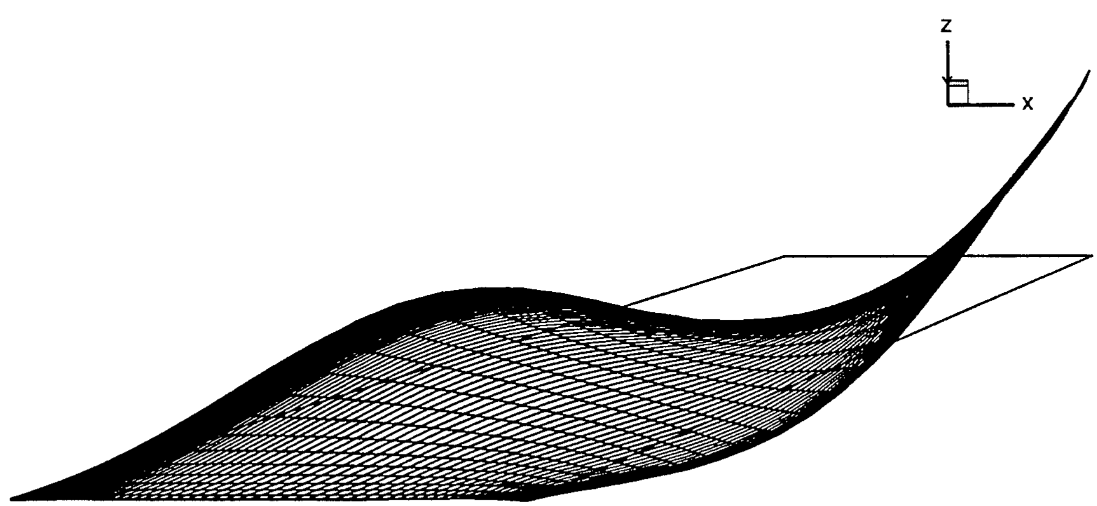


b) Mode Shape Deflection

Figure 13-3. Infinite-Plate Spline Results for AGARD 445 Wing, Mode Shape 3

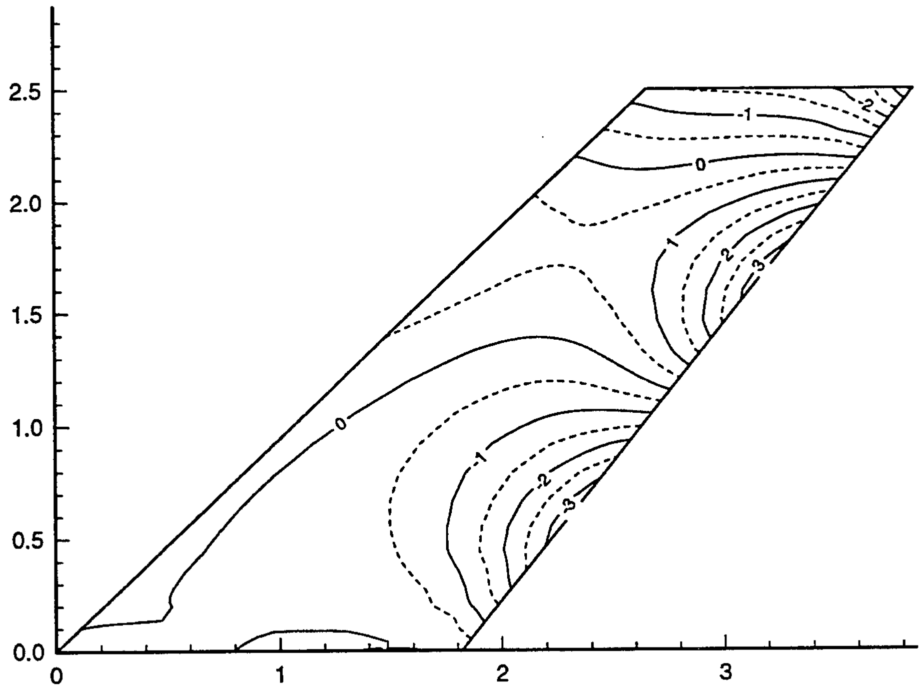


a) Mode Shape Contours

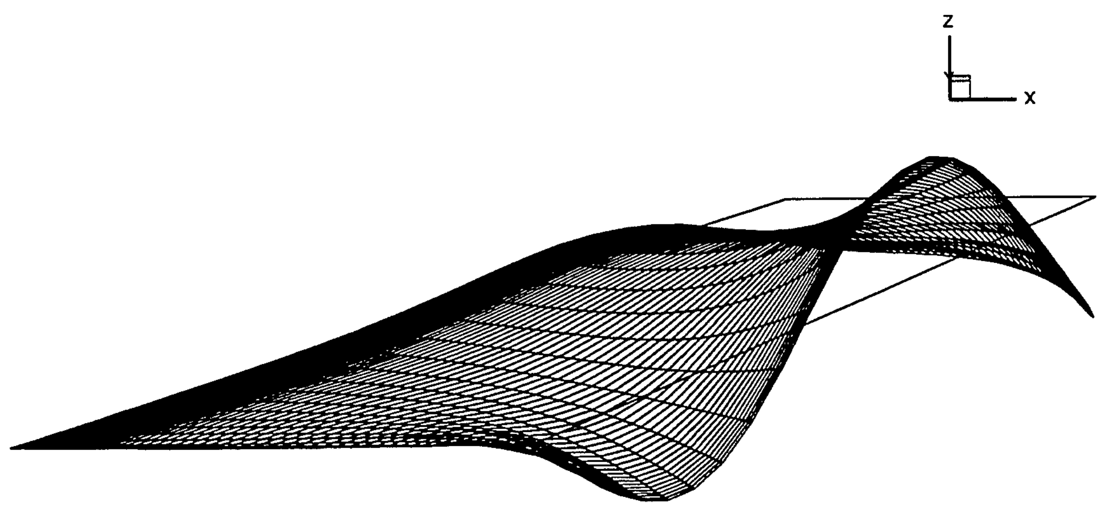


b) Mode Shape Deflection

Figure 13-4. Infinite-Plate Spline Results for AGARD 445 Wing, Mode Shape 4

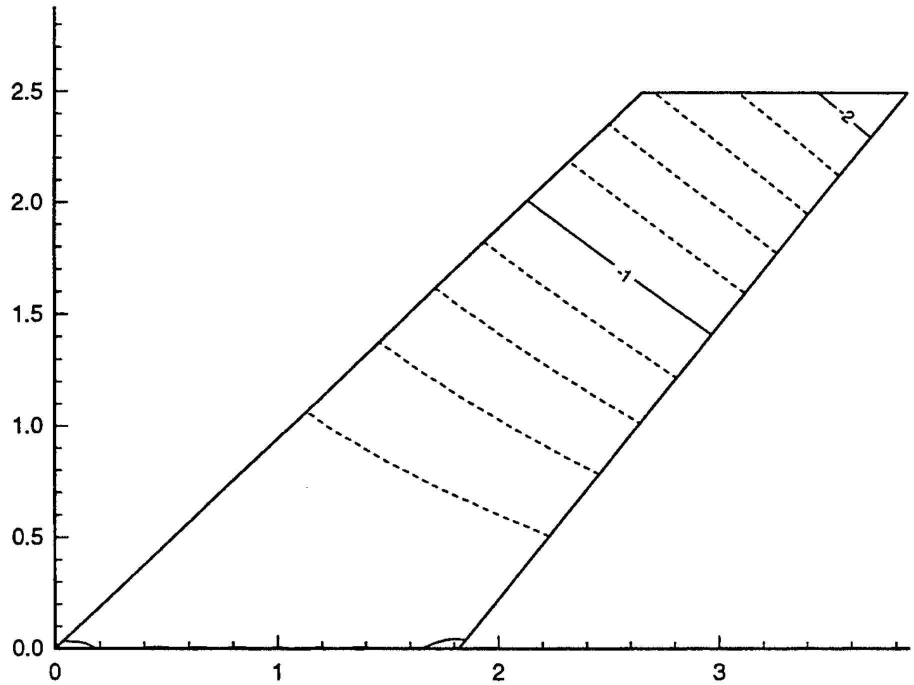


a) Mode Shape Contours

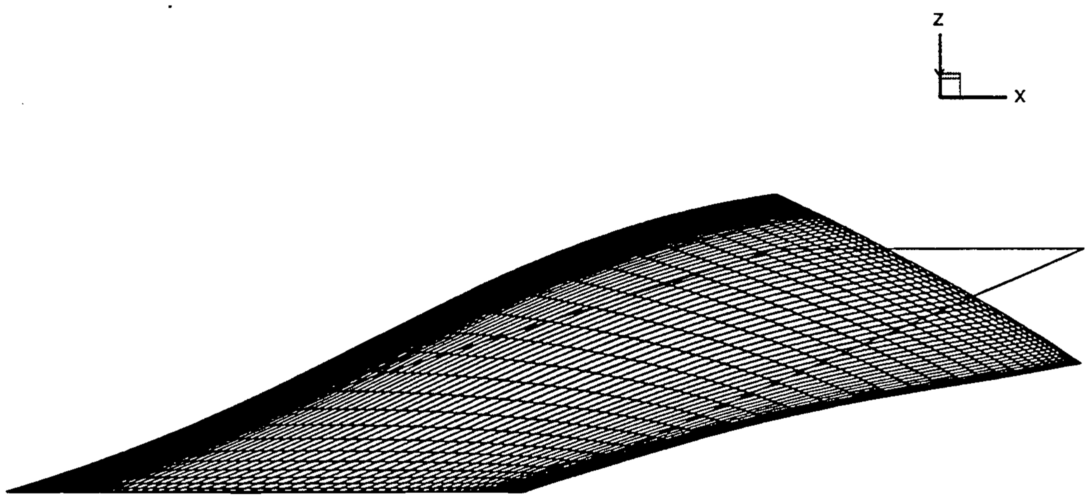


b) Mode Shape Deflection

Figure 13-5. Infinite-Plate Spline Results for AGARD 445 Wing, Mode Shape 5

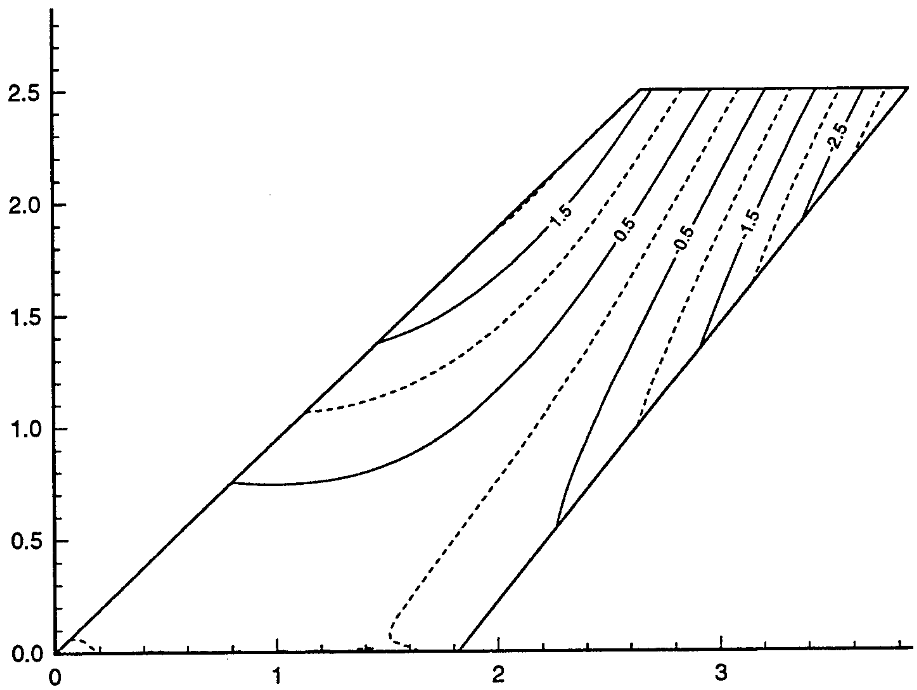


a) Mode Shape Contours

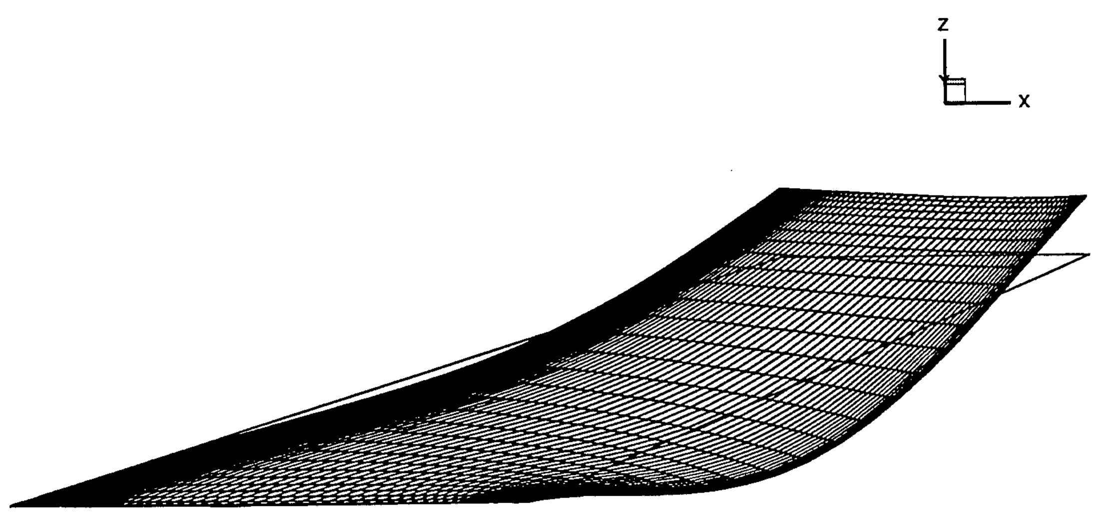


b) Mode Shape Deflection

Figure 13-6. Multiquadrics Results for AGARD 445 Wing, Mode Shape 1

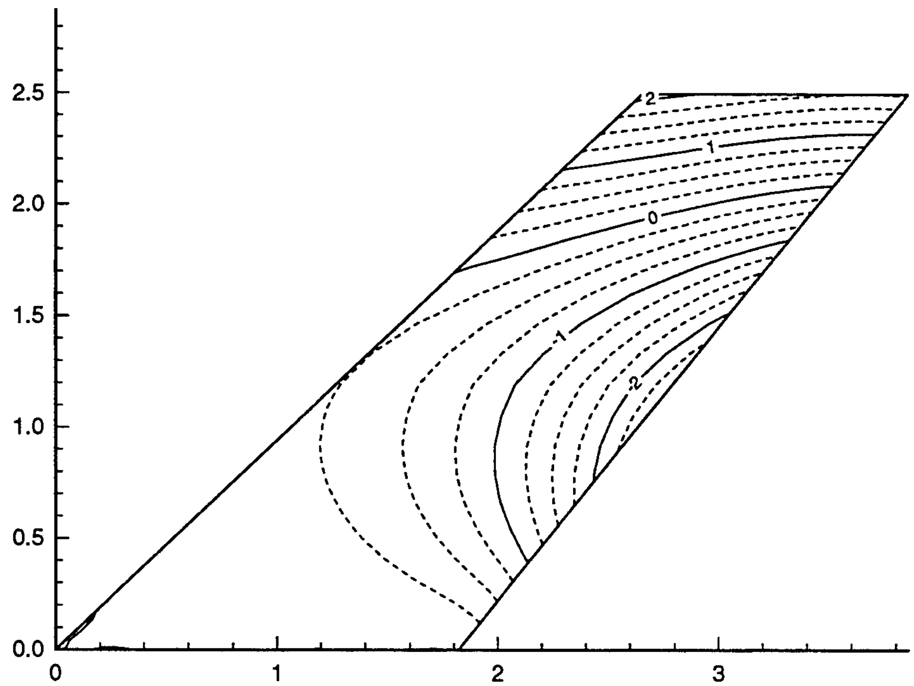


a) Mode Shape Contours

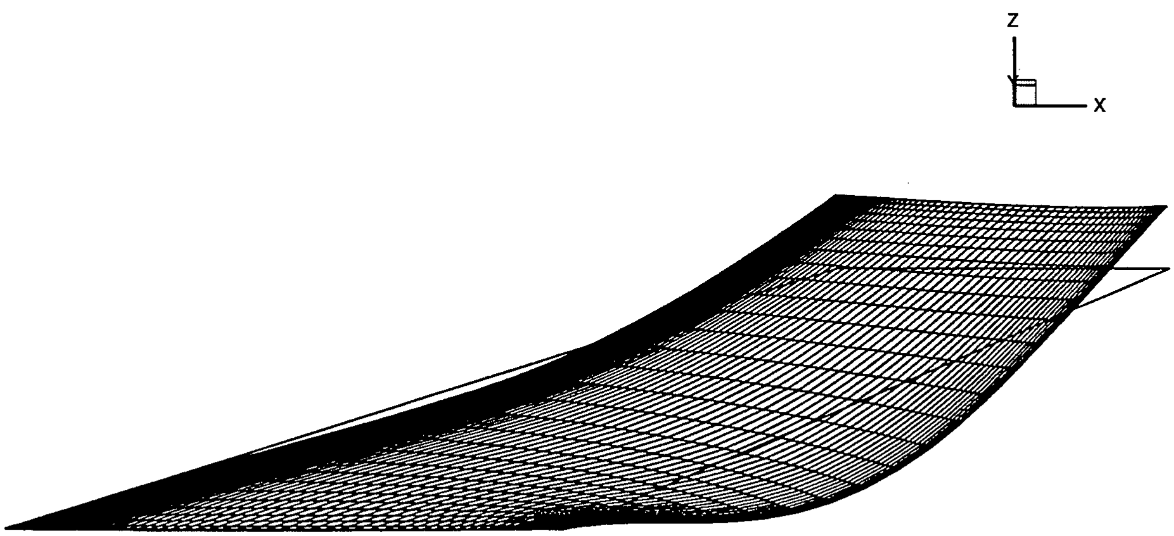


b) Mode Shape Deflection

Figure 13-7. Multiquadrics Results for AGARD 445 Wing, Mode Shape 2

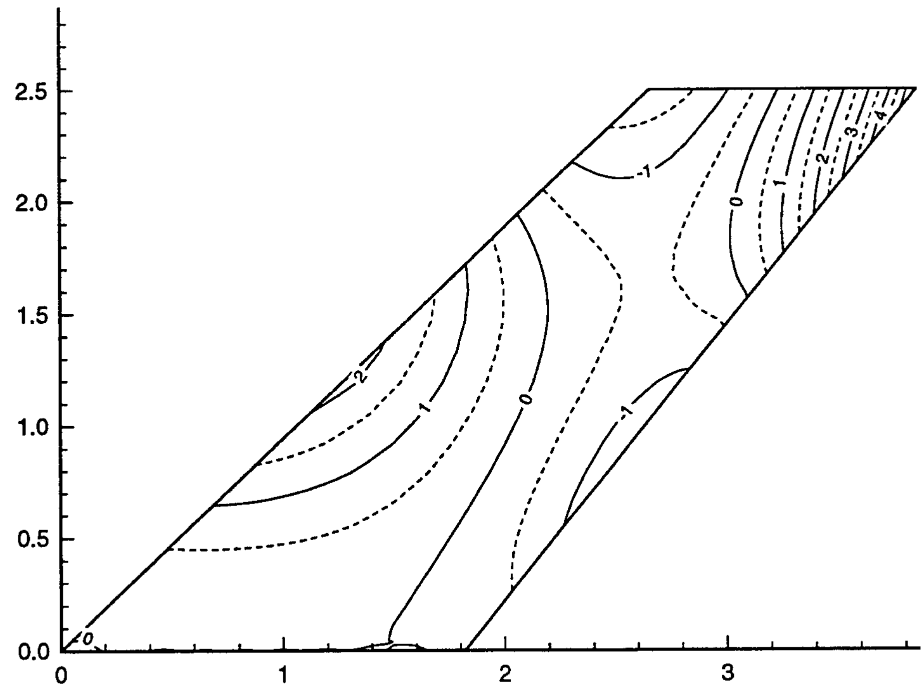


a) Mode Shape Contours

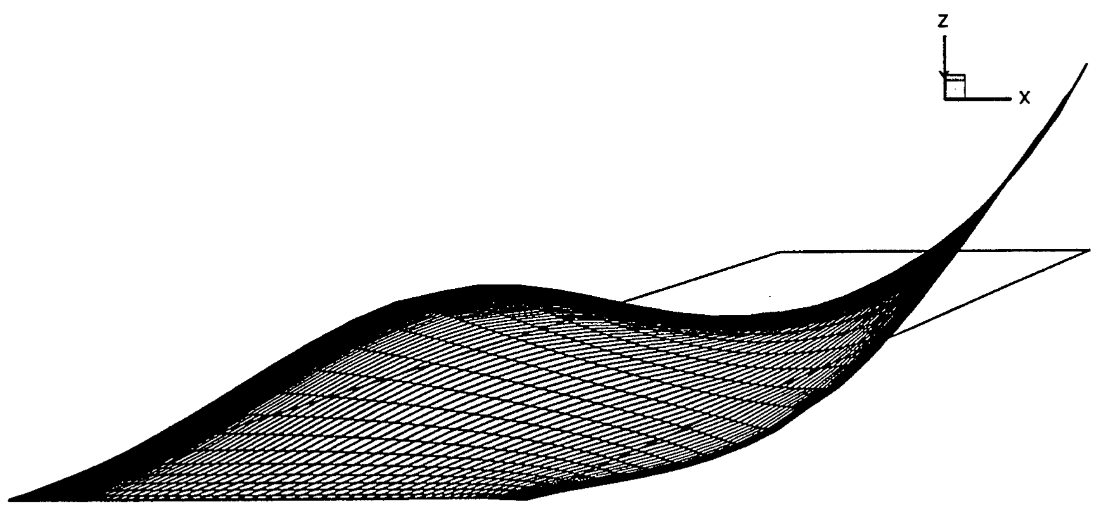


b) Mode Shape Deflection

Figure 13-8. Multiquadrics Results for AGARD 445 Wing, Mode Shape 3

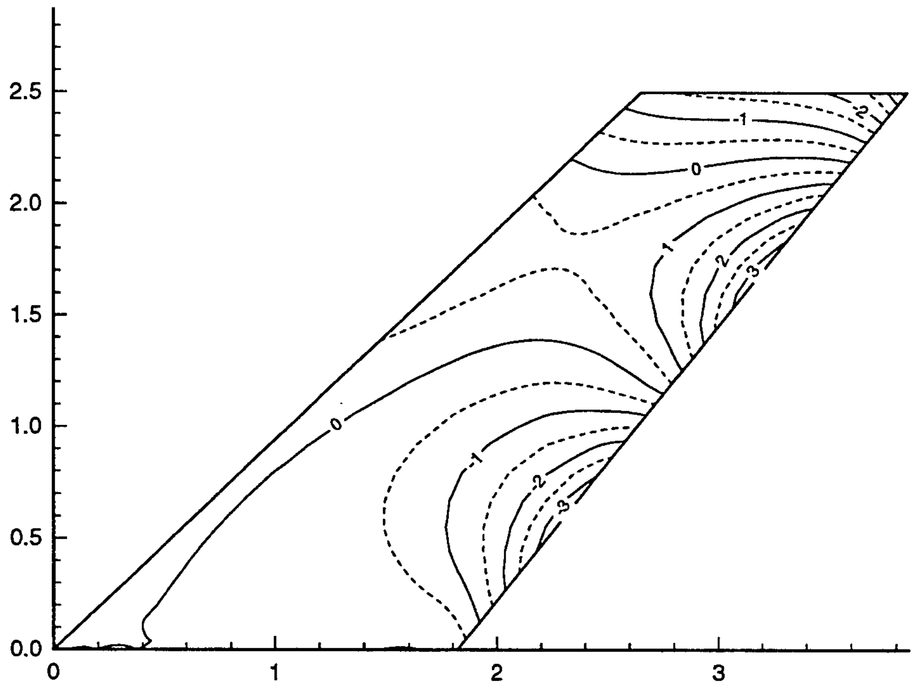


a) Mode Shape Contours

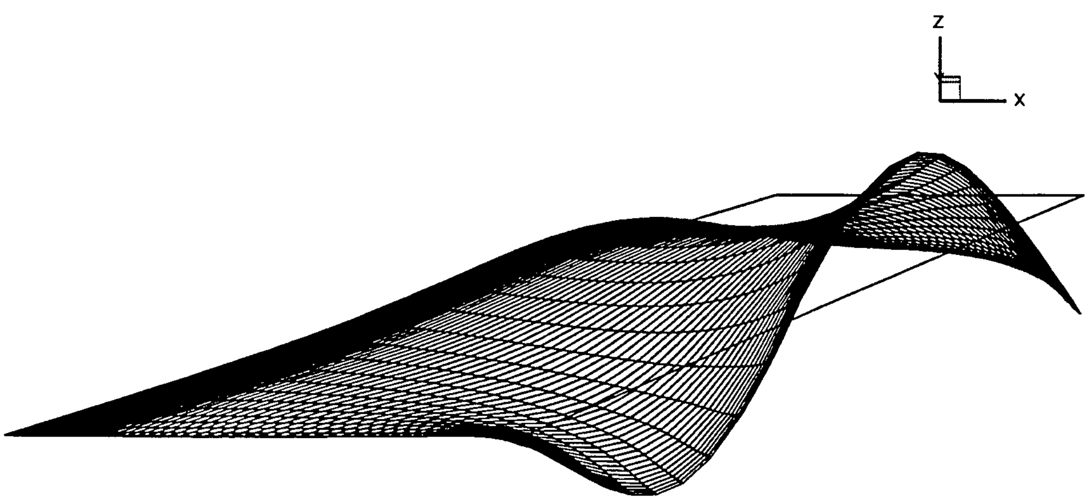


b) Mode Shape Deflection

Figure 13-9. Multiquadrics Results for AGARD 445 Wing, Mode Shape 4

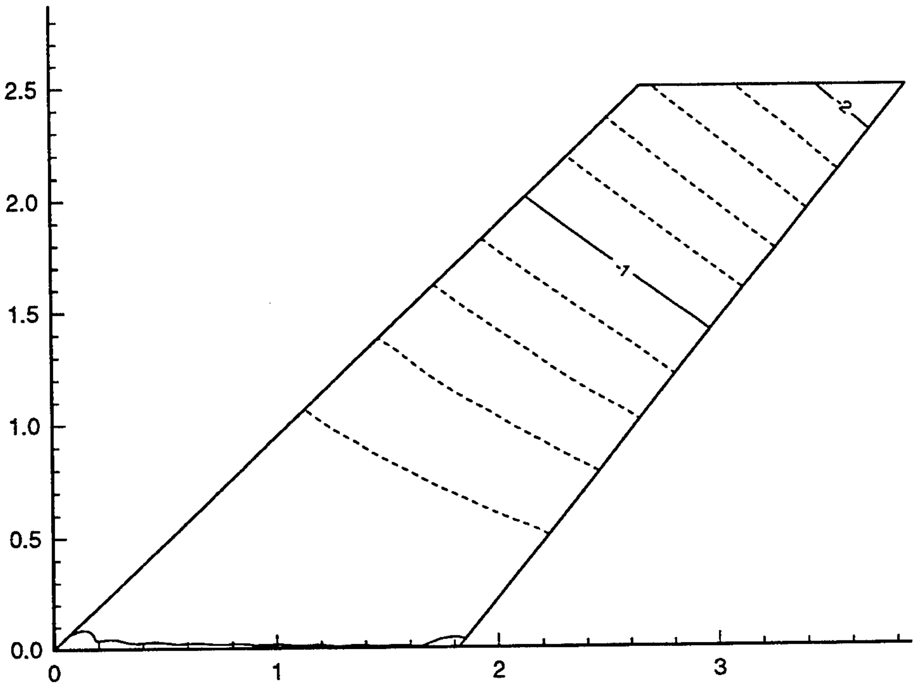


a) Mode Shape Contours

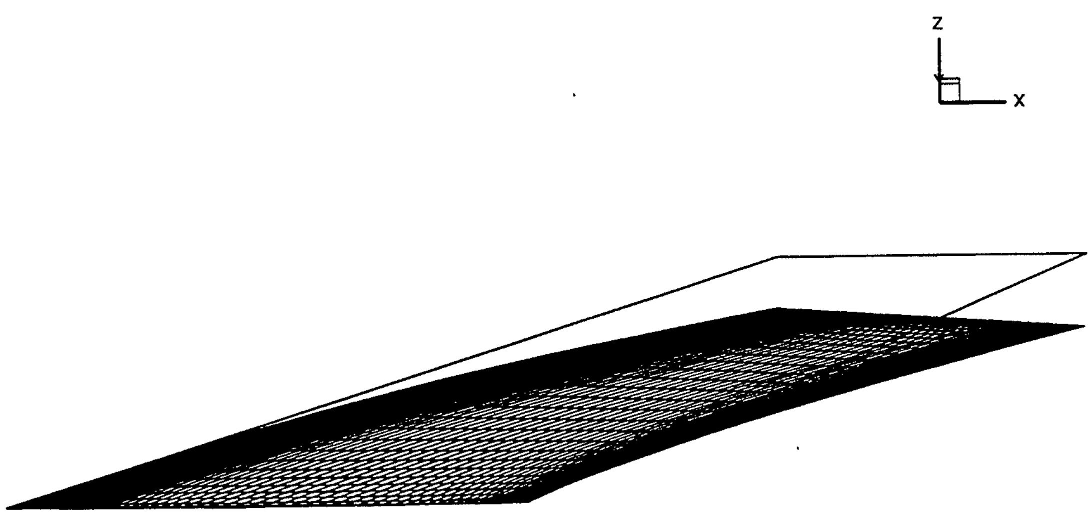


b) Mode Shape Deflection

Figure 13-10. Multiquadrics Results for AGARD 445 Wing, Mode Shape 5

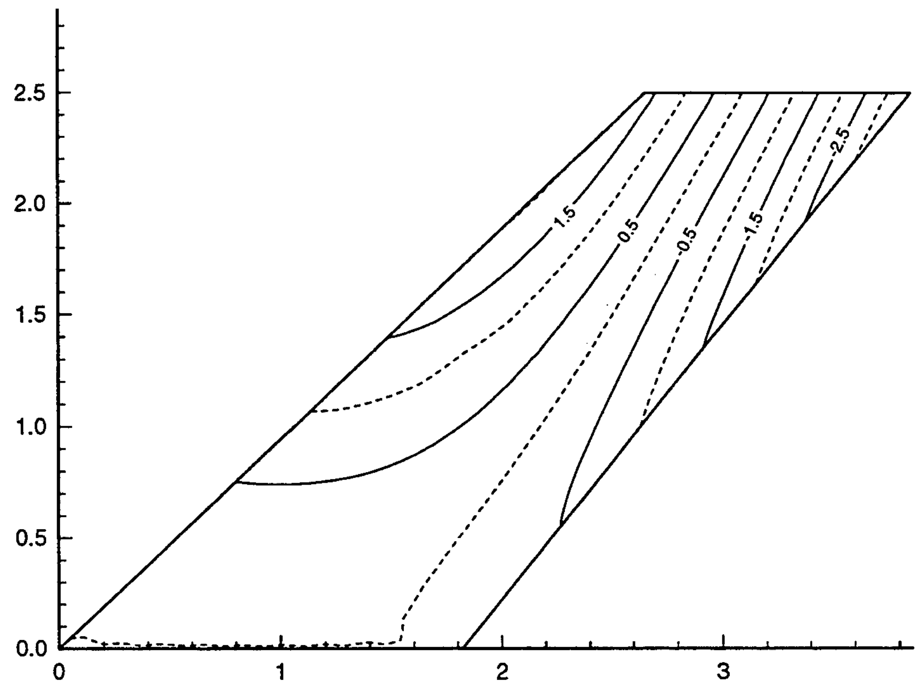


a) Mode Shape Contours

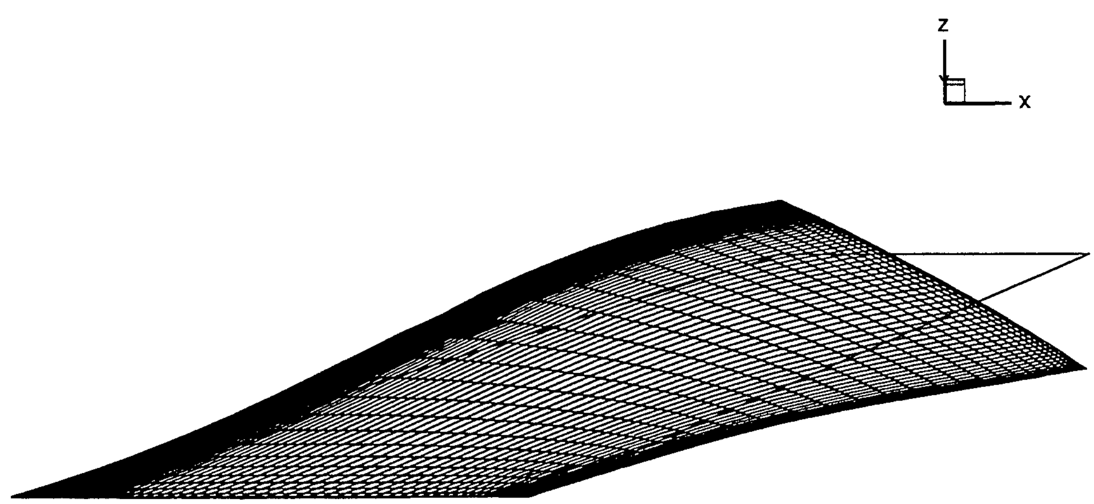


b) Mode Shape Deflection

Figure 13-11. Thin-Plate Spline Method Results for AGARD 445 Wing, Mode Shape 1

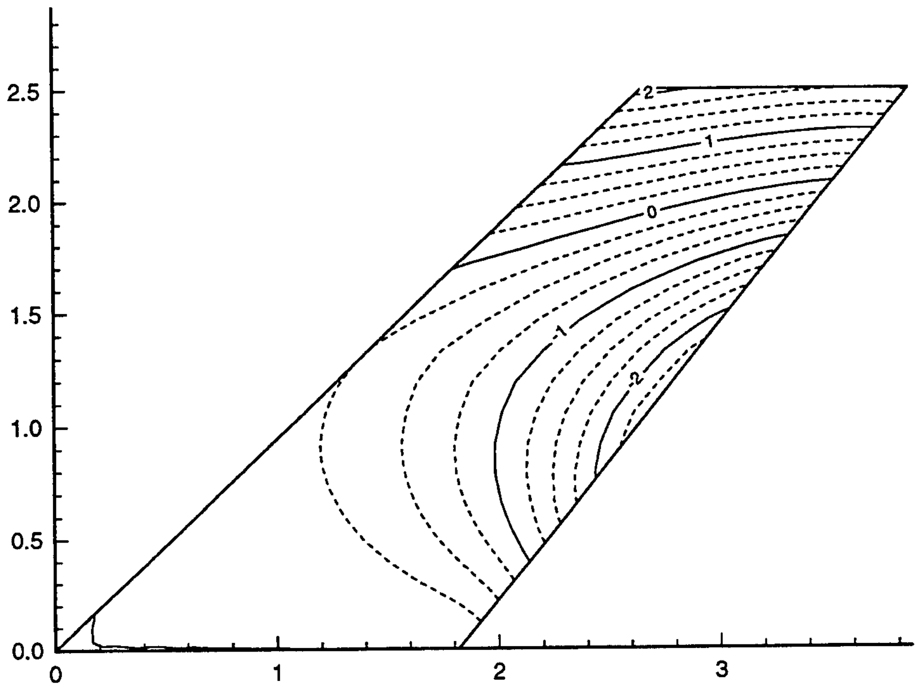


a) Mode Shape Contours

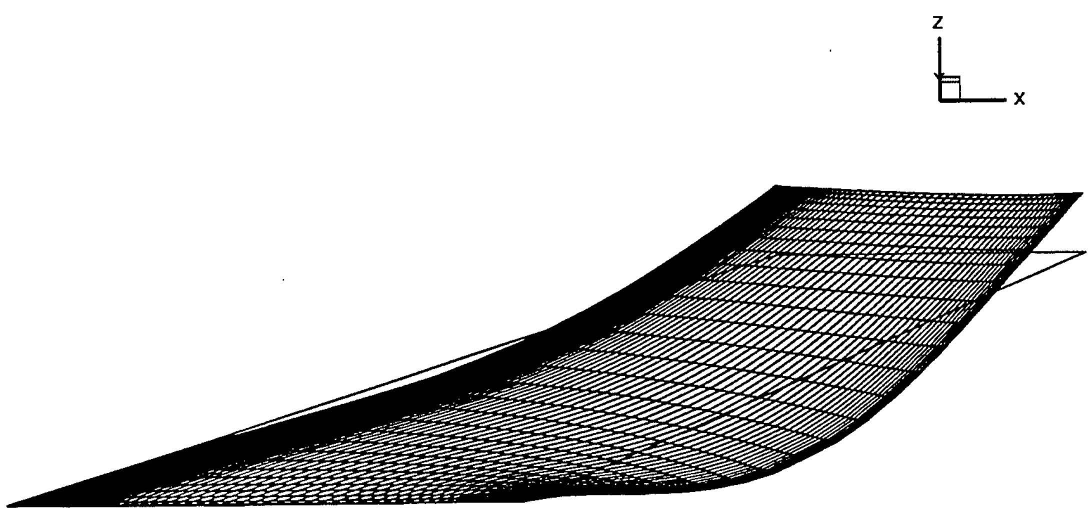


b) Mode Shape Deflection

Figure 13-12. Thin-Plate Spline Method Results for AGARD 445 Wing, Mode Shape 2

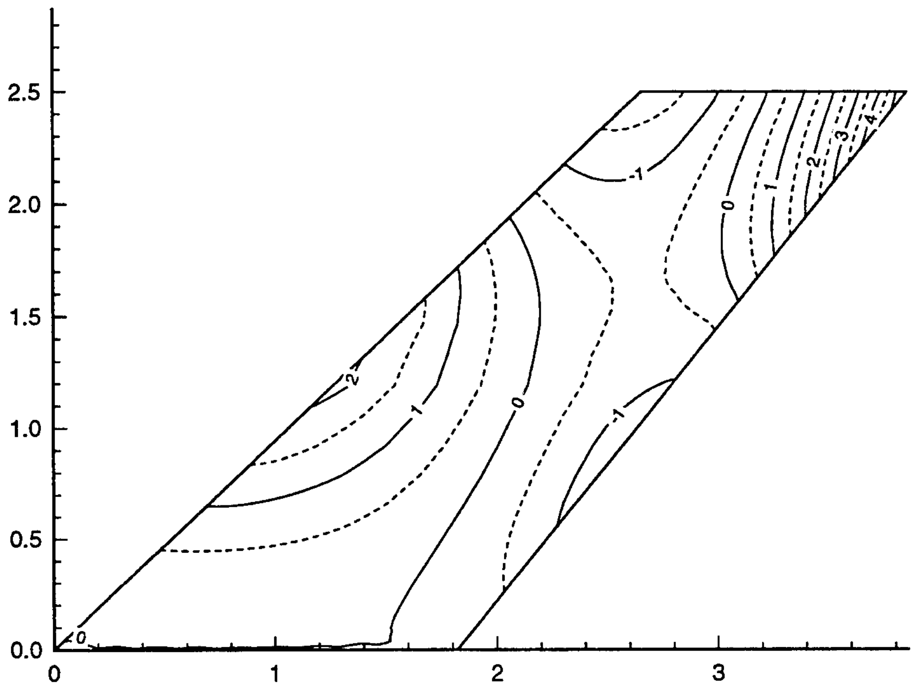


a) Mode Shape Contours

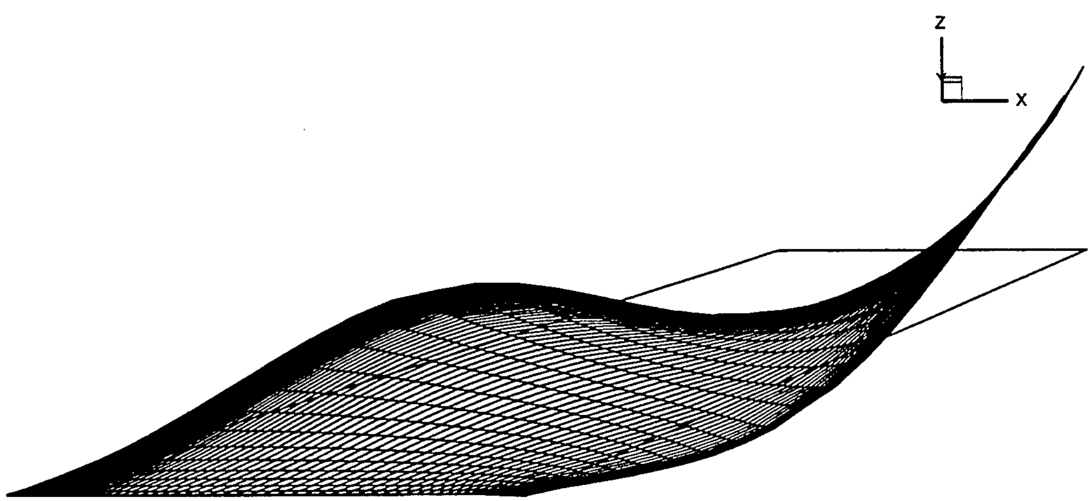


b) Mode Shape Deflection

Figure 13-13. Thin-Plate Spline Method Results for AGARD 445 Wing, Mode Shape 3

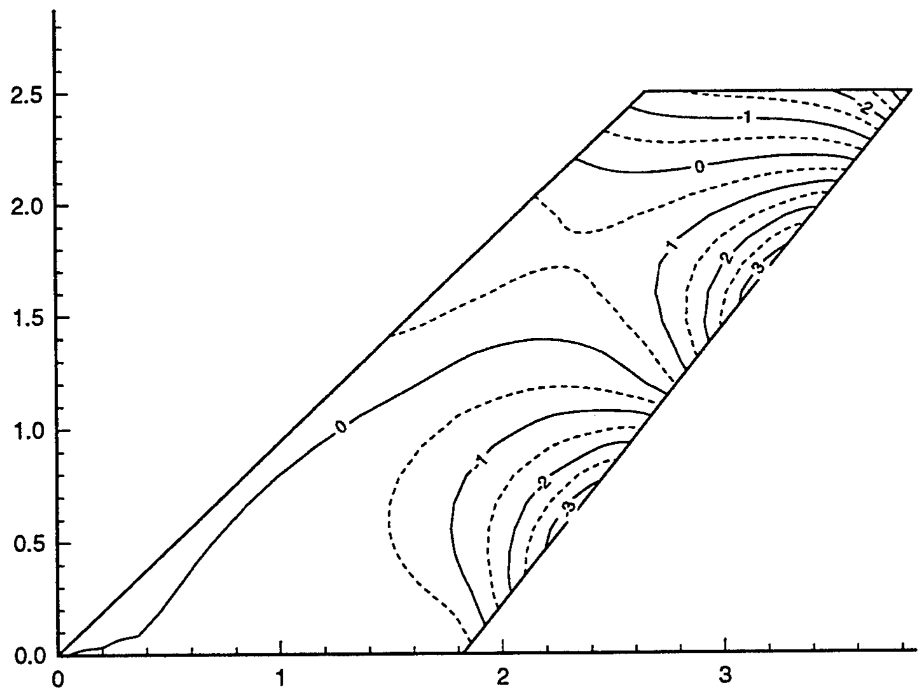


a) Mode Shape Contours

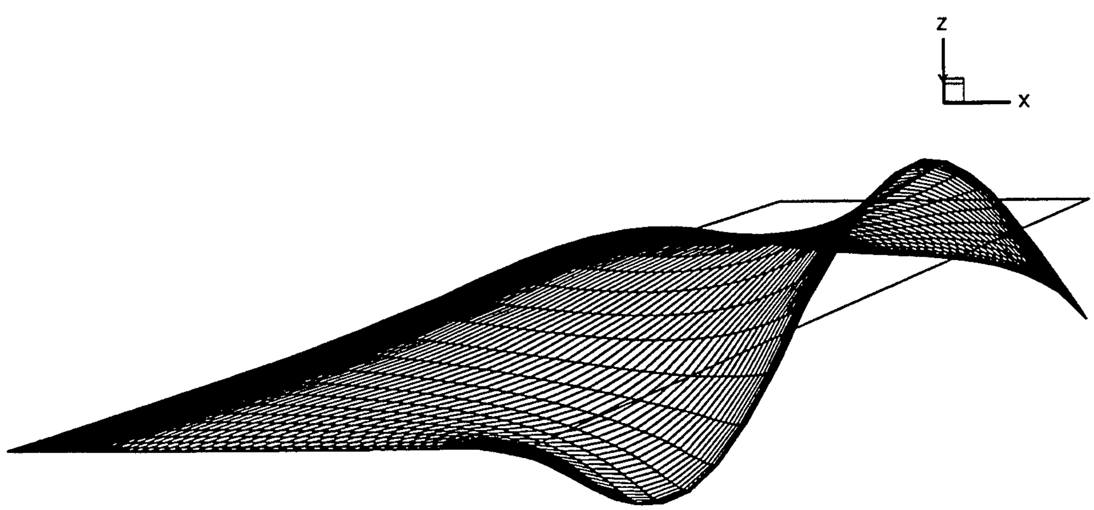


b) Mode Shape Deflection

Figure 13-14. Thin-Plate Spline Method Results for AGARD 445 Wing, Mode Shape 4

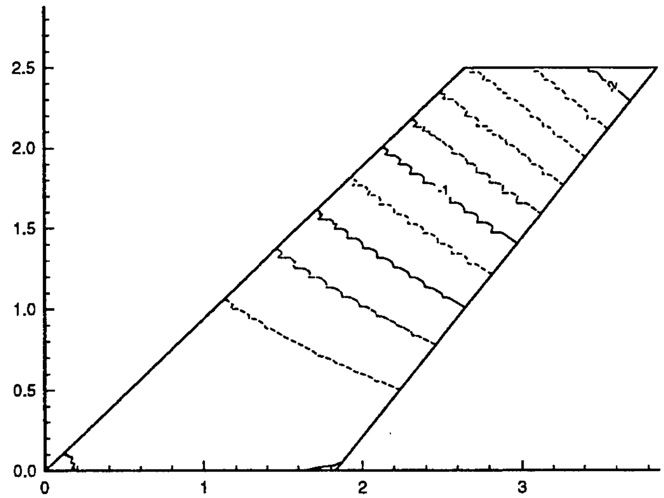


a) Mode Shape Contours



b) Mode Shape Deflection

Figure 13-15. Thin-Plate Spline Method Results for AGARD 445 Wing, Mode Shape 5



a) Mode Shape Contours

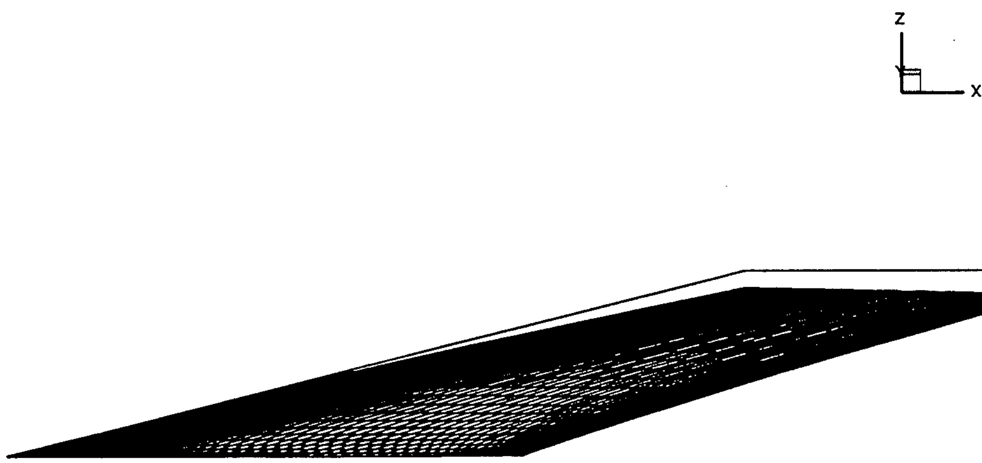
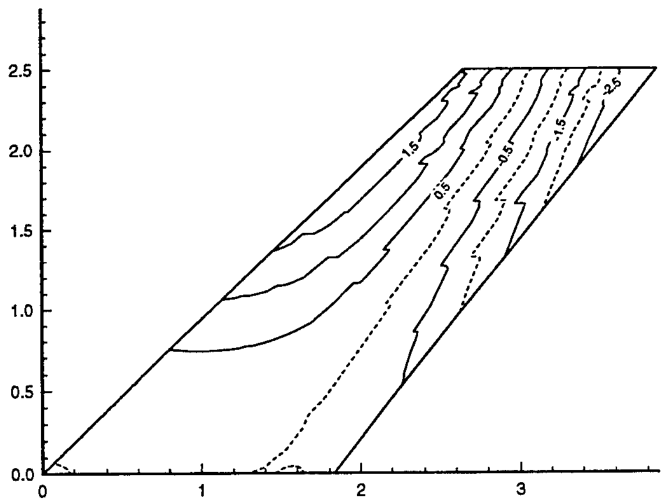


Figure 13-16. NUBS Results for AGARD 445 Wing, Mode Shape 1



a) Mode Shape Contours

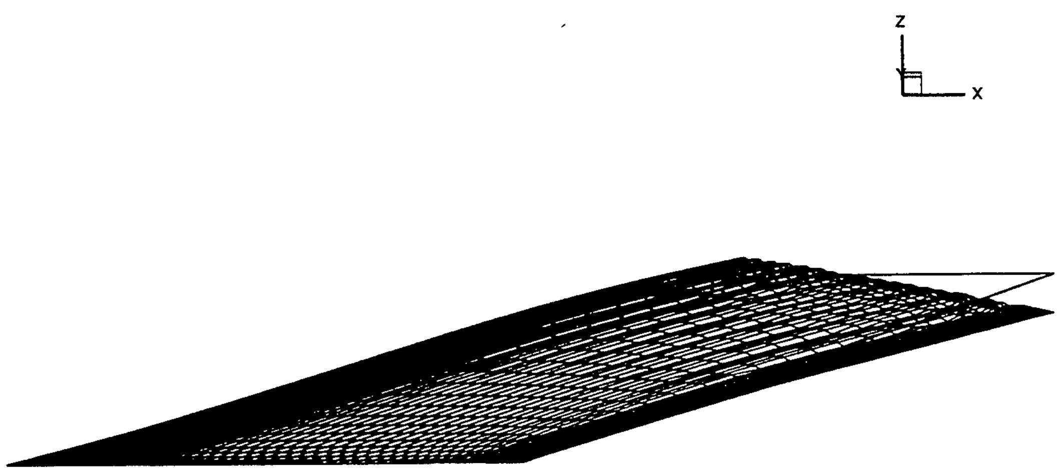
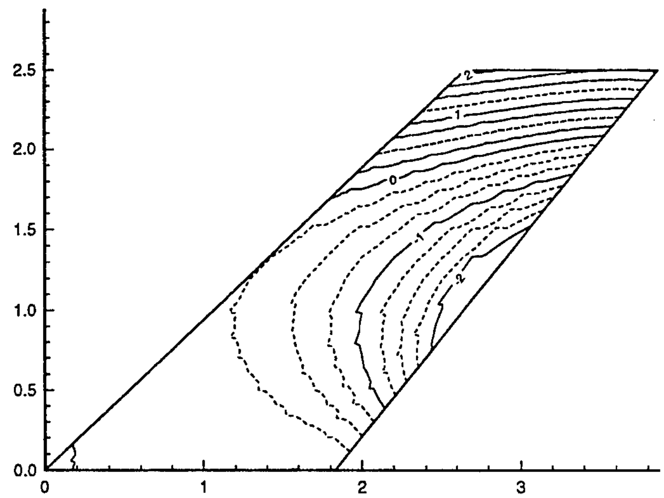


Figure 13-17. NUBS Results for AGARD 445 Wing, Mode Shape 2



a) Mode Shape Contours

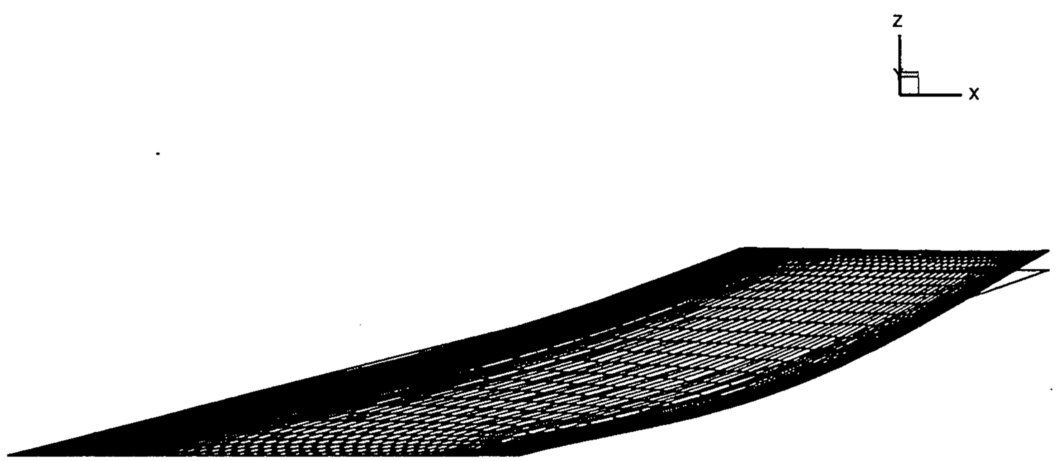
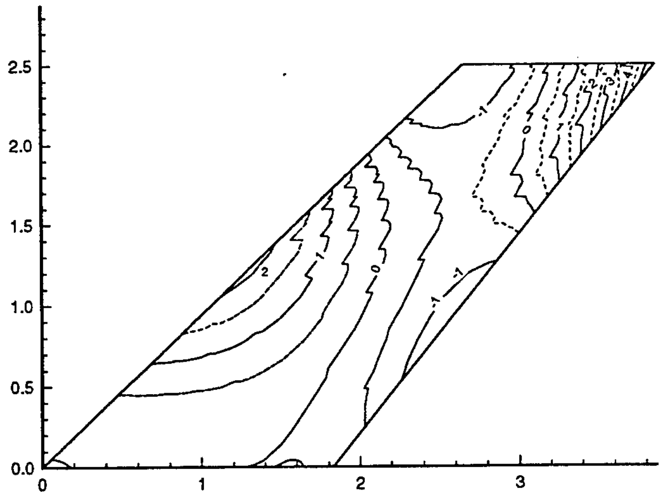


Figure 13-18. NUBS Results for AGARD 445 Wing, Mode Shape 3



a) Mode Shape Contours

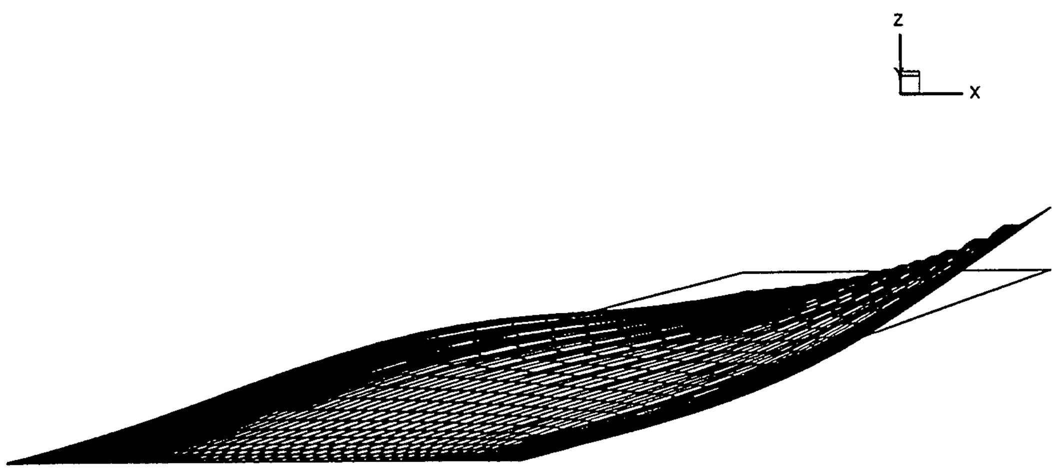
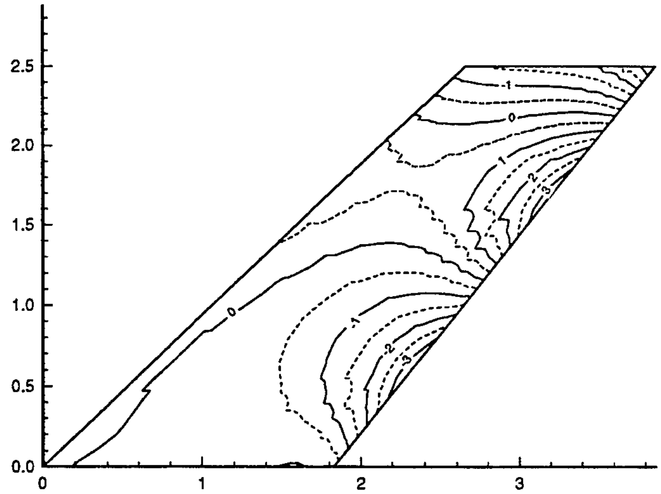


Figure 13-19. NUBS Results for AGARD 445 Wing, Mode Shape 4



a) Mode Shape Contours

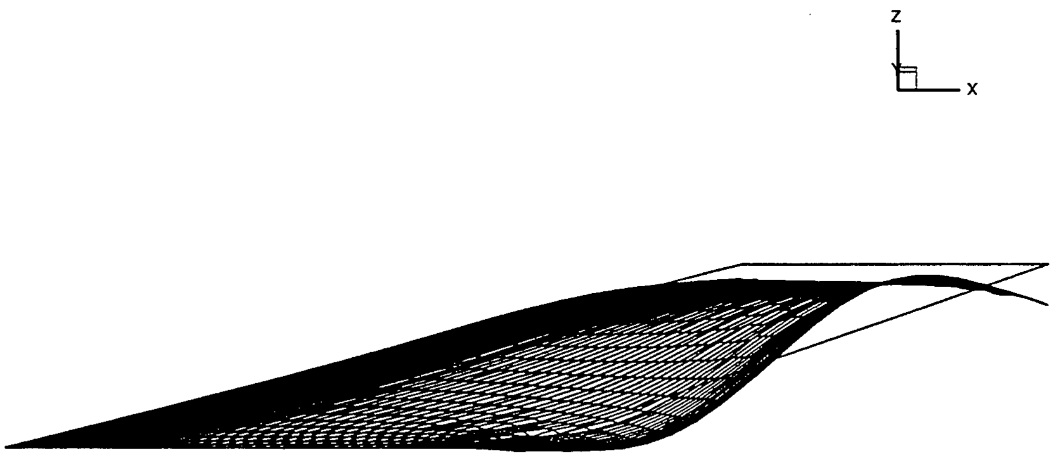
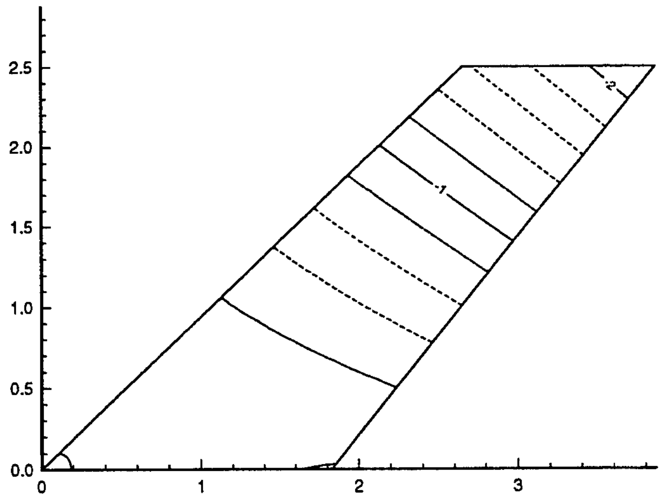
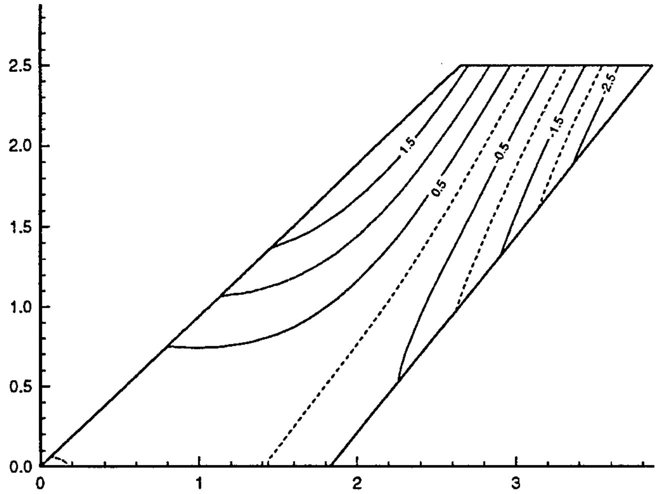


Figure 13-20. NUBS Results for AGARD 445 Wing, Mode Shape 5



a) Mode Shape Contours

Figure 13-21. Inverse Isoparametric Method Results for AGARD 445 Wing, Mode Shape 1



a) Mode Shape Contours

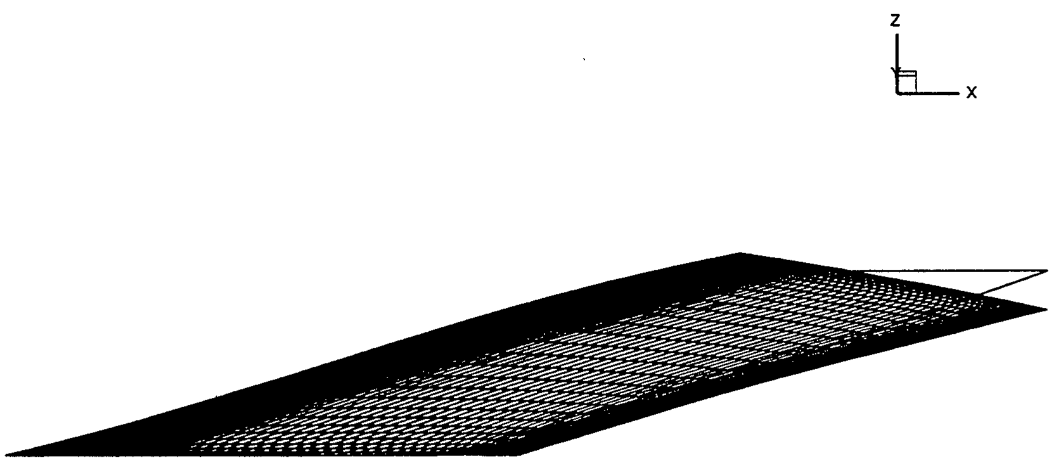
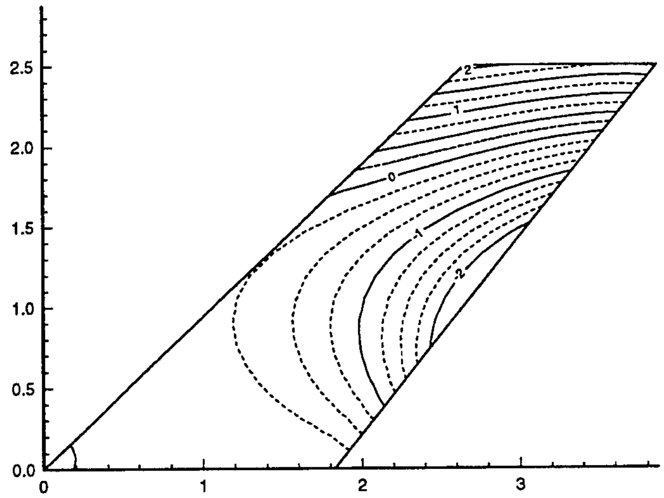


Figure 13-22. Inverse Isoparametric Method Results for AGARD 445 Wing, Mode Shape 2



a) Mode Shape Contours

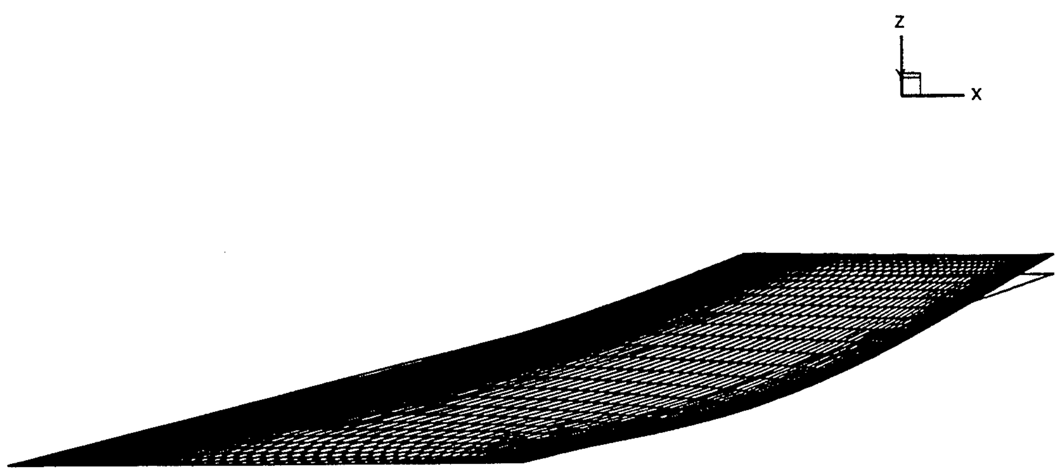
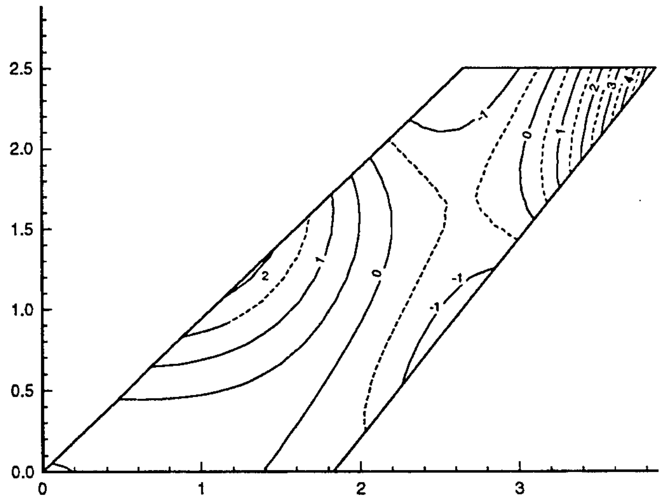


Figure 13-23. Inverse Isoparametric Method Results for AGARD 445 Wing, Mode Shape 3



a) Mode Shape Contours

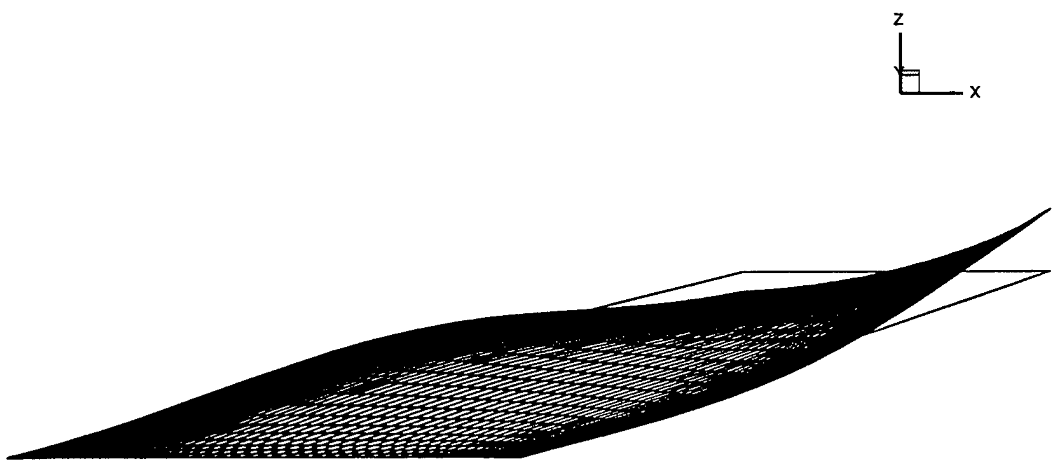
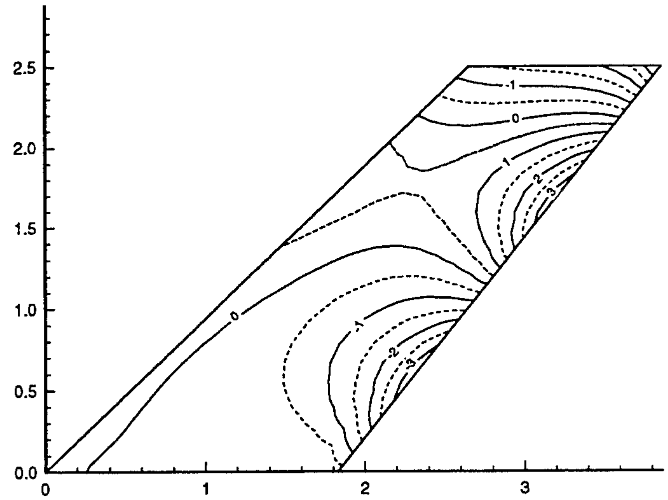


Figure 13-24. Inverse Isoparametric Method Results for AGARD 445 Wing, Mode Shape 4



a) Mode Shape Contours

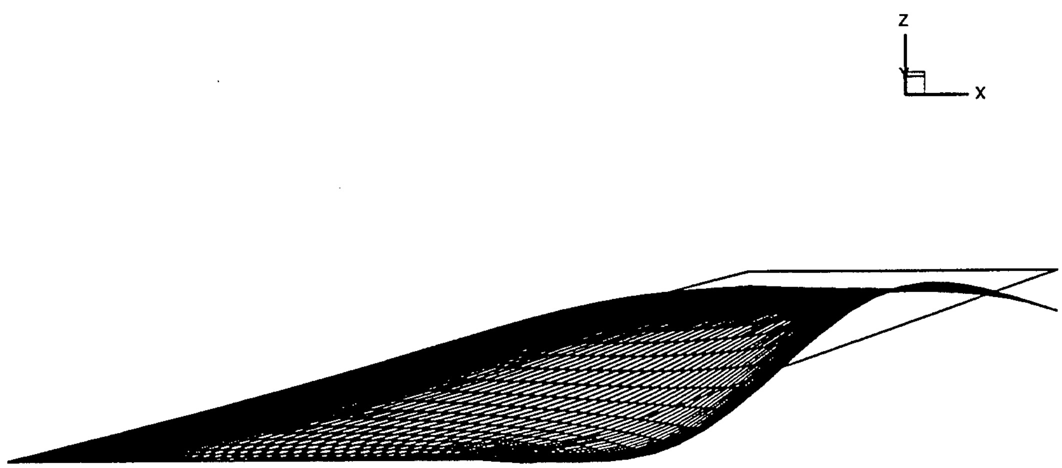


Figure 13-25. Inverse Isoparametric Method Results for AGARD 445 Wing, Mode Shape 5

13.2 Engine Liner Results

The engine liner mode shapes were interpolated using the IPS, TPS and MQ methods. The remaining methods failed for this particular configuration because of limitations cited in the previous chapters.

The results of the interpolations using these methods are given in Figures 13-26 to 13-30 for the IPS method, Figures 13-31 to 13-35 for the MQ method, Figures 13-36 to 13-40 for the TPS method. The format of these plots follow the format described in Chapter 12 for the engine liner problem.

The IPS method did not work for the engine liner configuration. It was necessary to divide the liner into four separate configurations, run them separately, then recombine the results to form Figures 13-26 to 13-30. Each section required approximately 30 minutes of workstation CPU time to complete (MQ and TPS required less than 5 minutes for the entire liner). If half or the full liner is run, IPS eventually fails with matrix solution errors after one hour or more. As seen in these figures, the interpolation appears to be excellent, with the exception of the break points in the liner. The mis-matching breakpoints mean that the integrity of the endpoints of the configuration are not being maintained.

It is evident that MQ has some problems with the engine liner. In Figures 13-31 to 13-35, the superimposed mode shapes have very irregular oscillations at the top and bottom of the engine liner. When the multiquadrics method is scaled or run using different *r* values, the problem shifts slightly, but does not go away. The interpolation by TPS shows excellent results IF the method is scaled. The mode deflections match almost identically to the original data. For the unscaled run, TPS has similar problems to the multiquadrics methods.

Table 13.2 Maximum Deflections For the Engine Liner Mode Shapes

Mode Shape	Original	Thin-Plate Spline (Unscaled)	Thin-Plate Spline (Scaled)	Multiquadrics
1	5.737	123.1	5.713	131.8
2	5.801	82.75	5.775	27.06
3	5.797	197.	5.770	84.25
4	5.611	28.03	5.670	14.54
5	5.676	26.65	5.653	157.3

* IPS values were not available

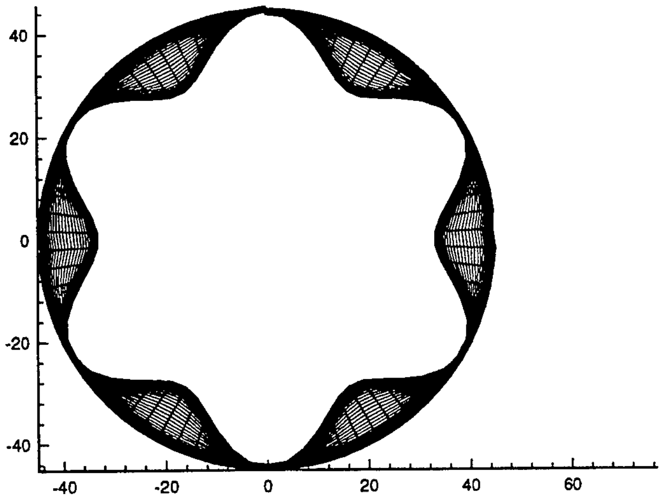


Figure 13-26. Engine Liner Mode Shape 1 Using Infinite-Plate Spline Method

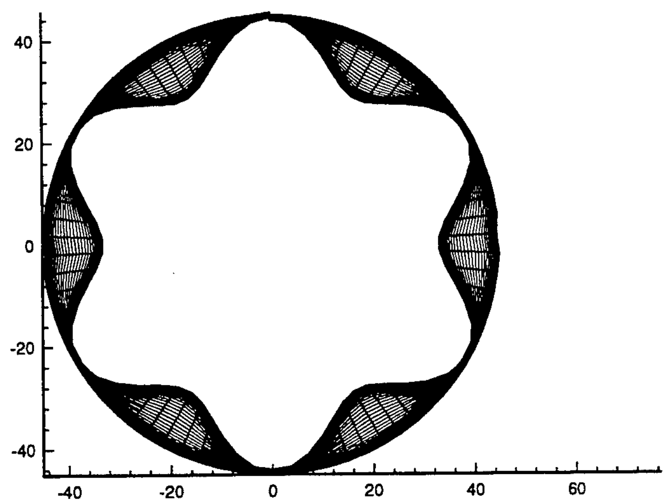


Figure 13-27. Engine Liner Mode Shape 2 Using Infinite-Plate Spline Method

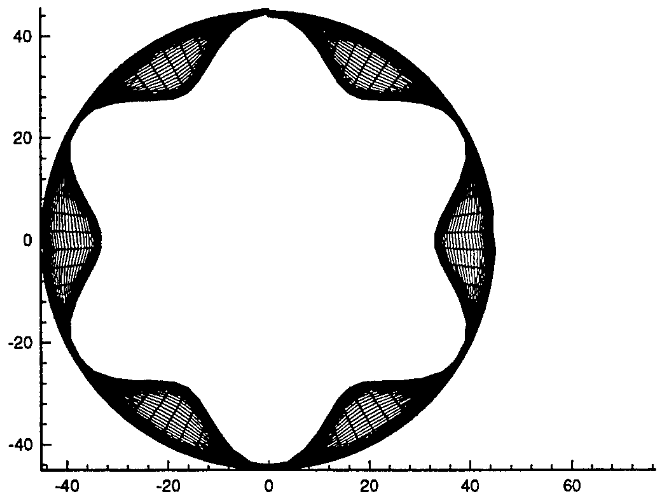


Figure 13-28. Engine Liner Mode Shape 3 Using Infinite-Plate Spline Method

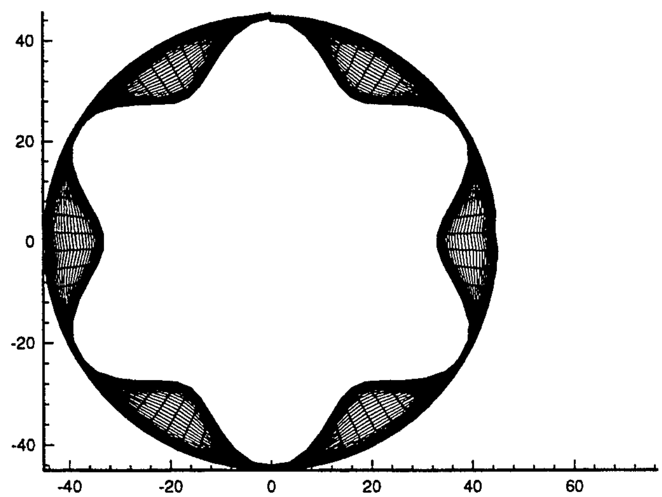


Figure 13-29. Engine Liner Mode Shape 4 Using Infinite-Plate Spline Method

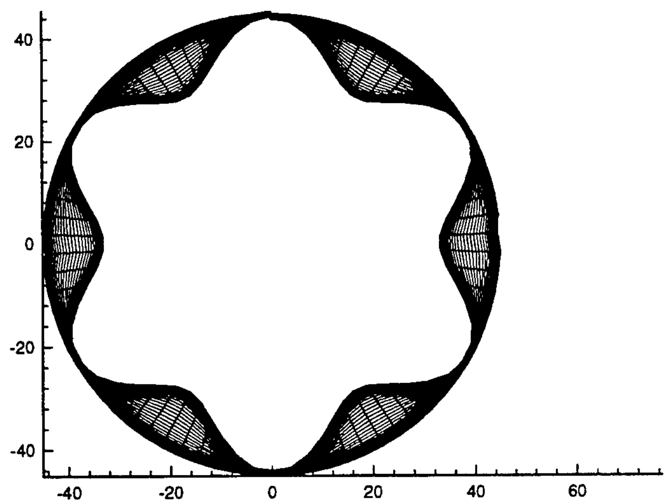


Figure 13-30. Engine Liner Mode Shape 5 Using Infinite-Plate Spline Method

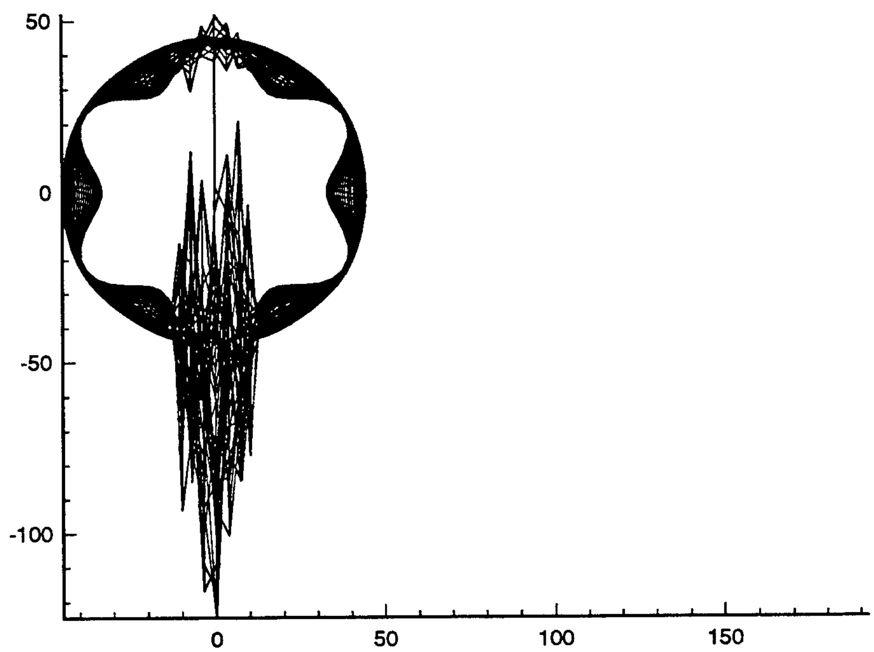


Figure 13-31. Engine Liner Mode Shape 1 Using Multiquadrics Method

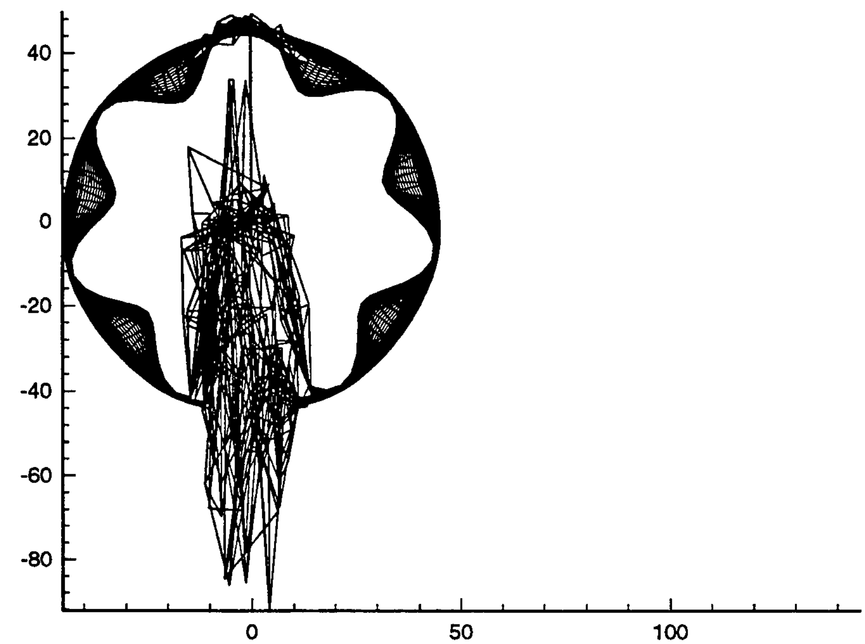


Figure 13-32. Engine Liner Mode Shape 2 Using Multiquadrics Method

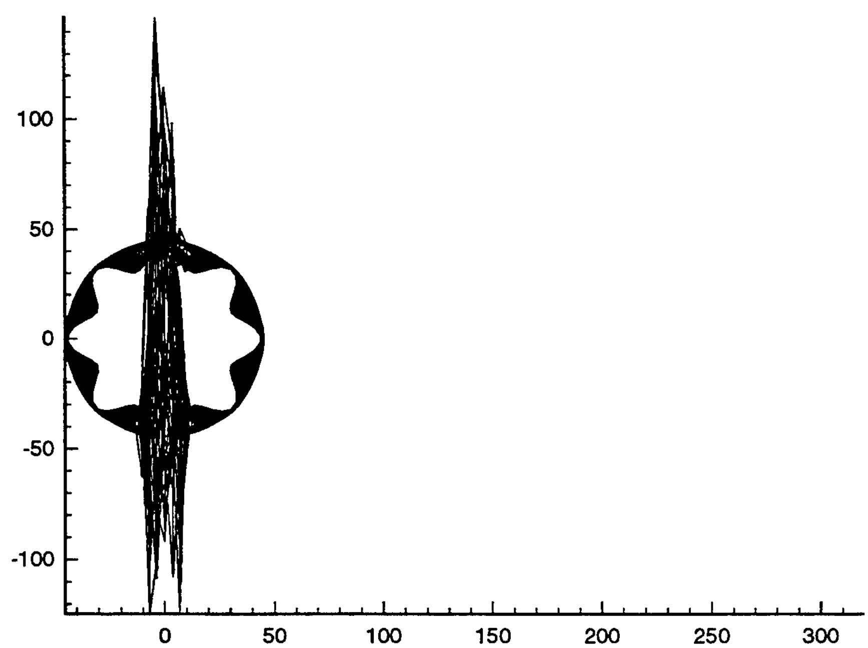


Figure 13-33. Engine Liner Mode Shape 3 Using Multiquadrics Method

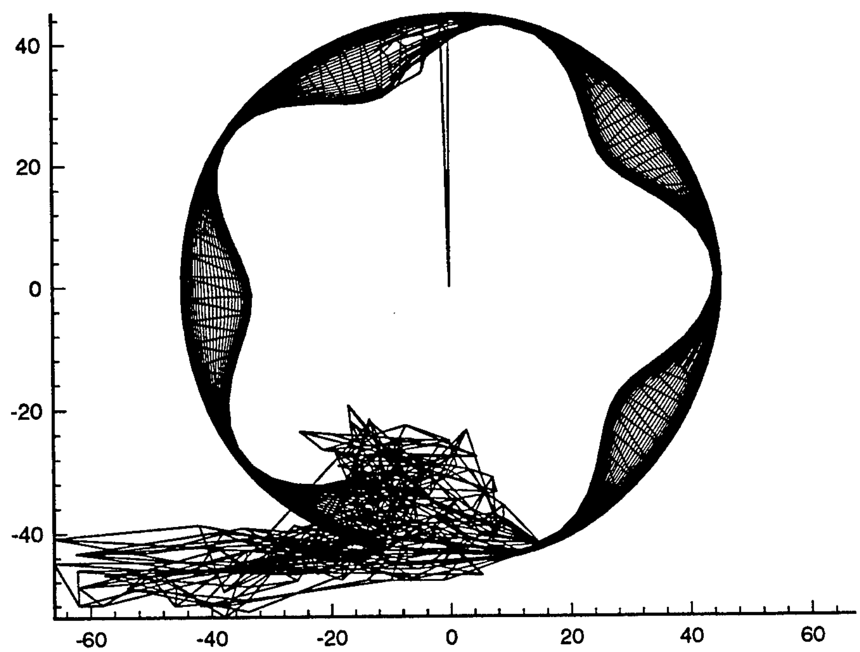


Figure 13-34. Engine Liner Mode Shape 4 Using Multiquadrics Method

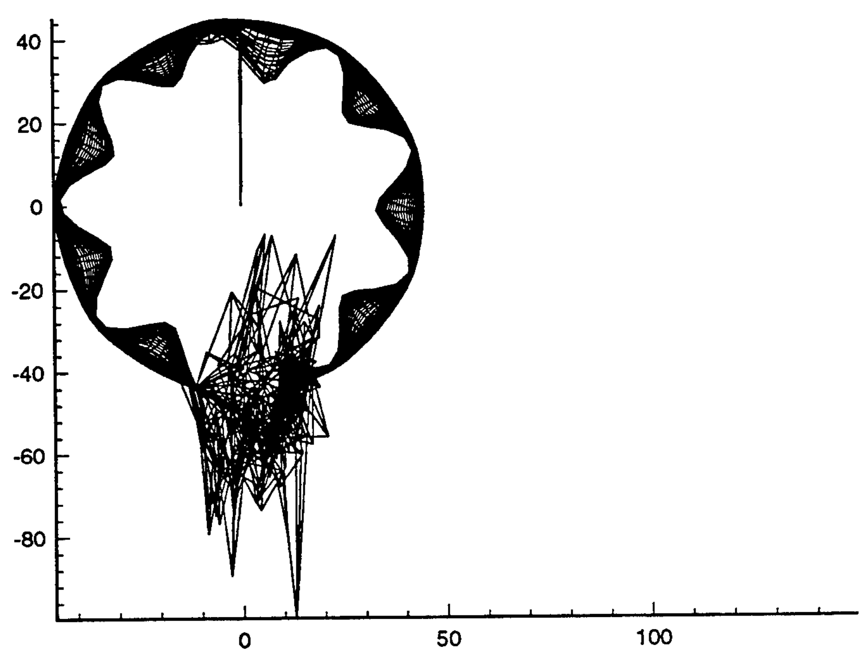


Figure 13-35. Engine Liner Mode Shape 5 Using Multiquadrics Method

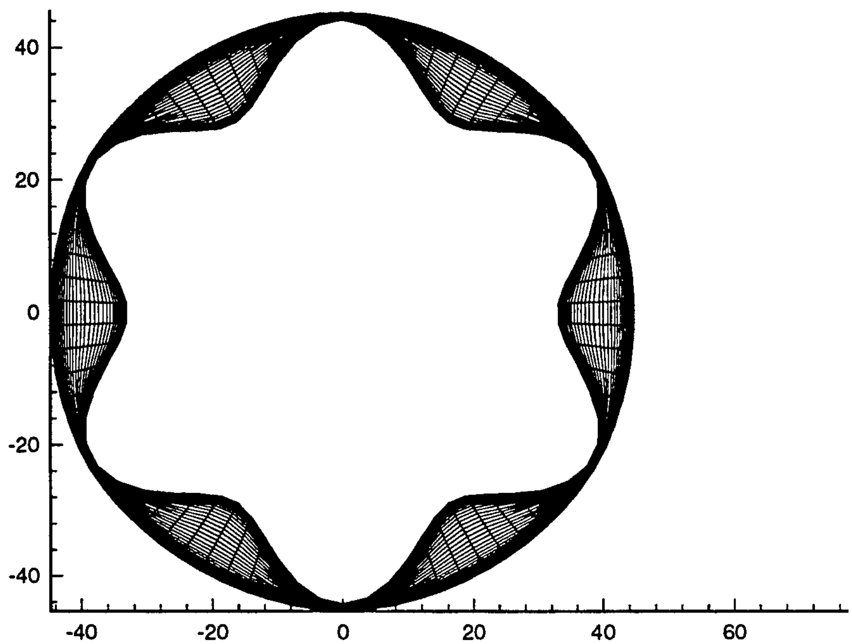


Figure 13-36. Engine Liner Mode Shape 1 Using Thin-Plate Spline Method (Scaled)

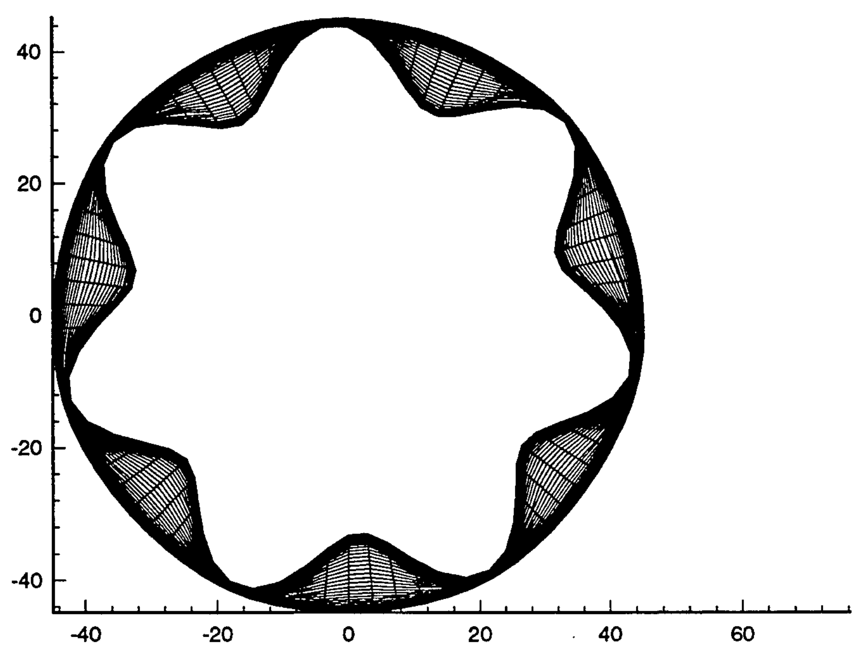


Figure 13-37. Engine Liner Mode Shape 2 Using Thin-Plate Spline Method (Scaled)

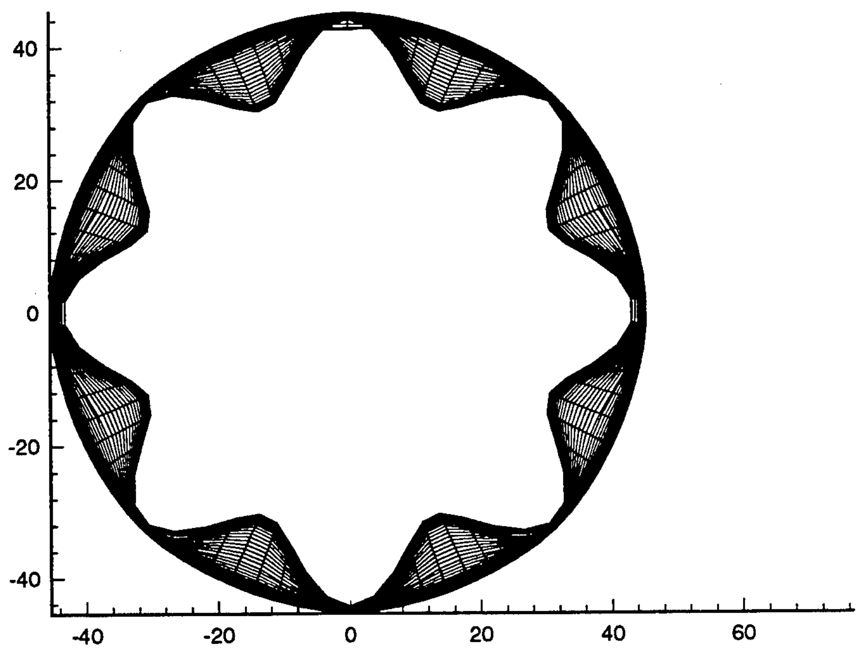


Figure 13-38. Engine Liner Mode Shape 3 Using Thin-Plate Spline Method (Scaled)

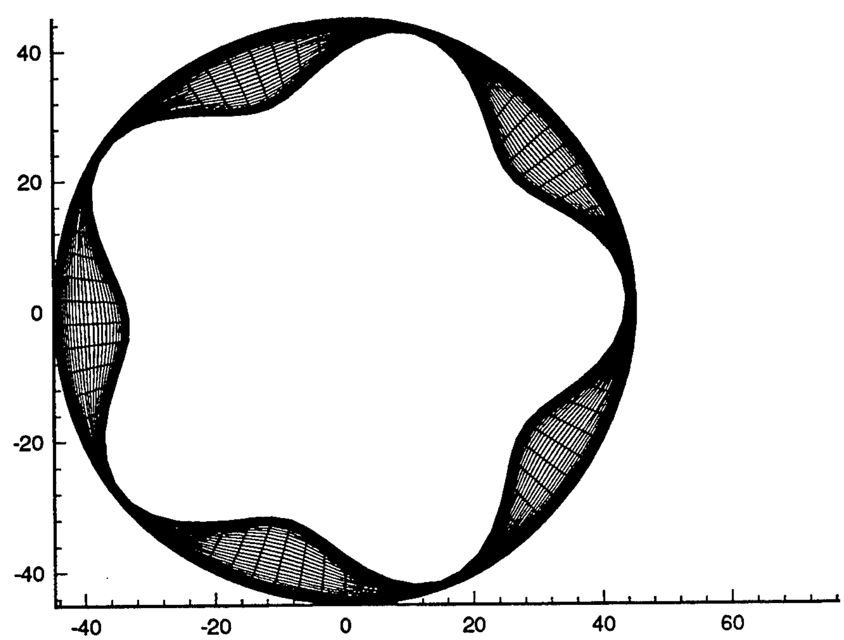


Figure 13-39. Engine Liner Mode Shape 4 Using Thin-Plate Spline Method (Scaled)

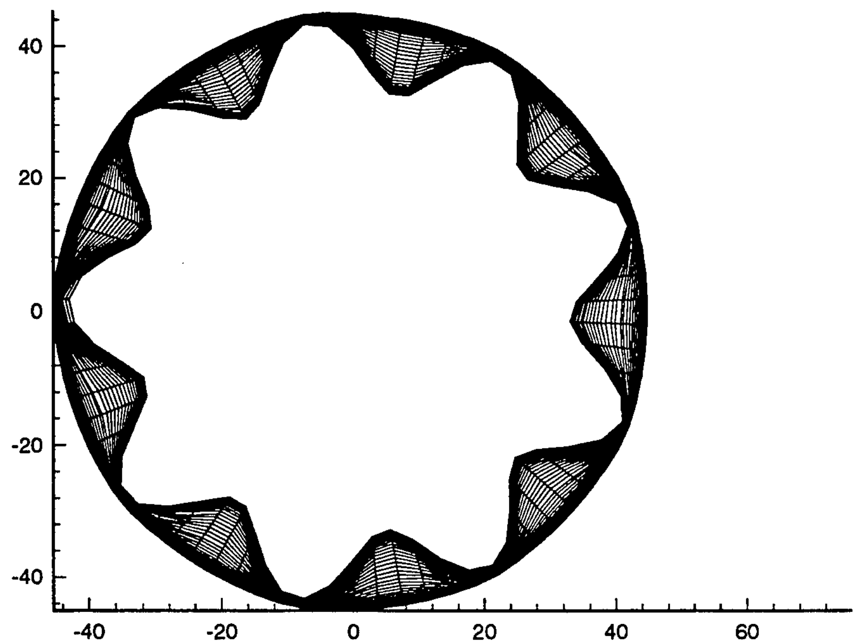


Figure 13-40. Engine Liner Mode Shape 5 Using Thin-Plate Spline Method (Scaled)

13.3 Generic Hypersonic Vehicle Results

The generic hypersonic vehicle mode shapes were interpolated using the multiquadrics (MQ), Infinite-Plate Spline (IPS) and Thin-Plate Spline (TPS) methods. The remaining methods were excluded because of previously discussed limitations for this particular configuration.

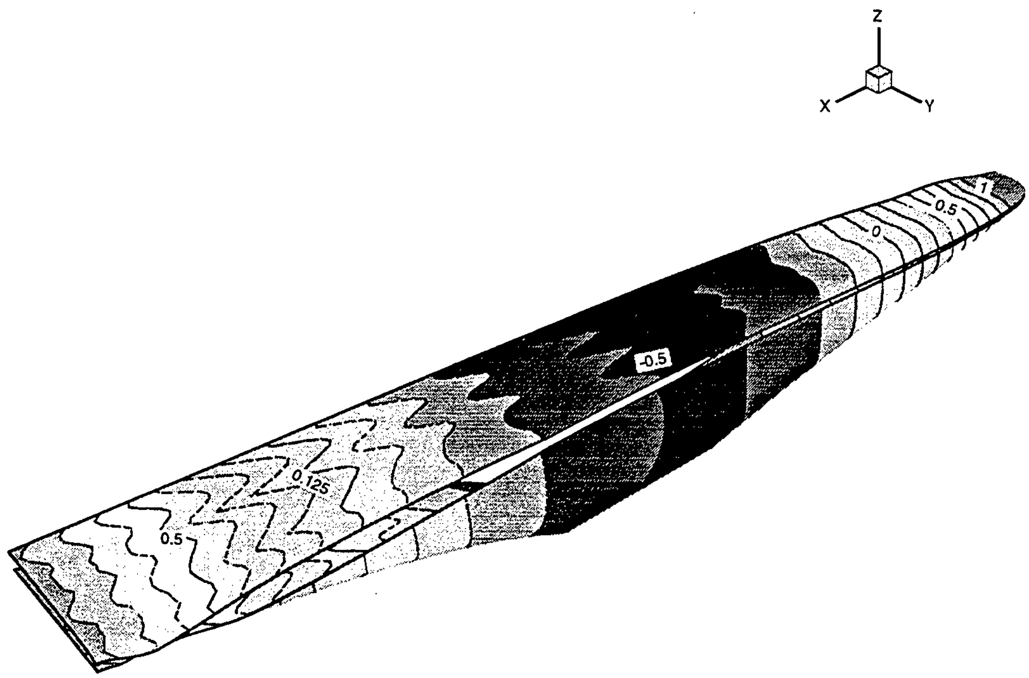
The results of the these methods are given in Figures 13-41 – 13-47 for the IPS method, Figures 13-48 – 13-54 for the MQ method, and Figures 13-55 – 13-61 for the TPS method. The format of these plots follow the format described in Chapter 12, for the generic hypersonic vehicle problem (Section 12.3).

From the statistical summary of the problem in Table 13.3, it appears that the methods provide equivalent results. However, a comparison of the results in the figures shows that IPS introduces some oscillations into the results that are not present originally or in the MQ and TPS results.

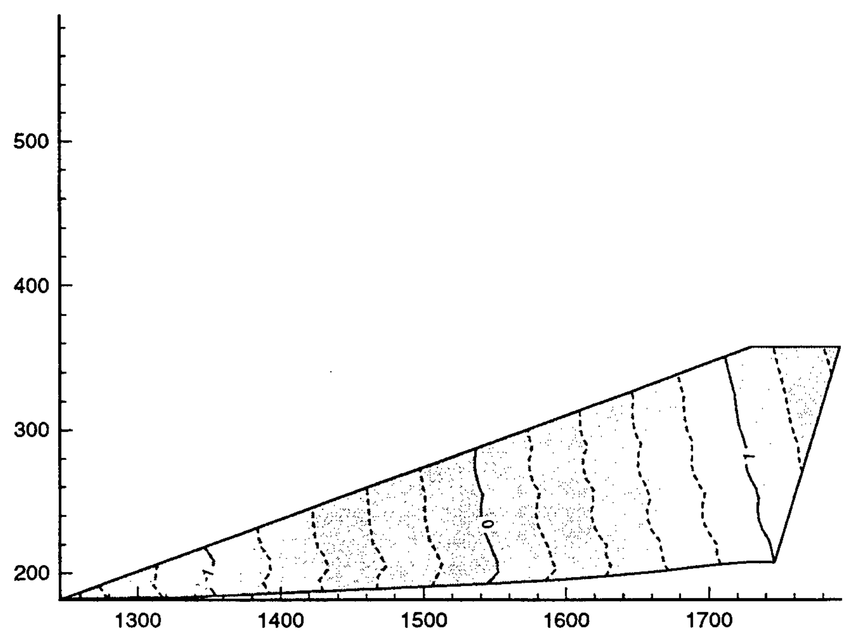
The MQ and TPS methods give excellent results IF they are not scaled. Recall that the engine liner problem of Sections 12.2 and 13.2 required that the multiquadrics and thin-plate spline methods be scaled in order to provide reasonable results. Here, the numerical summary (as shown in Table 13.3) does not indicate a problem. However, Figure 13-62 shows the interpolation of mode shape 1 if scaling is not used. Similar results are obtained on the remaining six mode shapes as well. It is evident that an additional parameter needs to be added to the statistical summaries to check for oscillatory results.

Table 13.3 Maximum Deflections For the Generic Hypersonic Mode Shape 1

Component	Original	IPS	TPS (Scaled & Unscaled)	MQ (Scaled & Unscaled)
Fuselage	1.405	44354	1.415	1.411
Upper Wing	1.490	1.579	1.491	1.570
Lower Wing	1.490	1.580	1.491	1.569

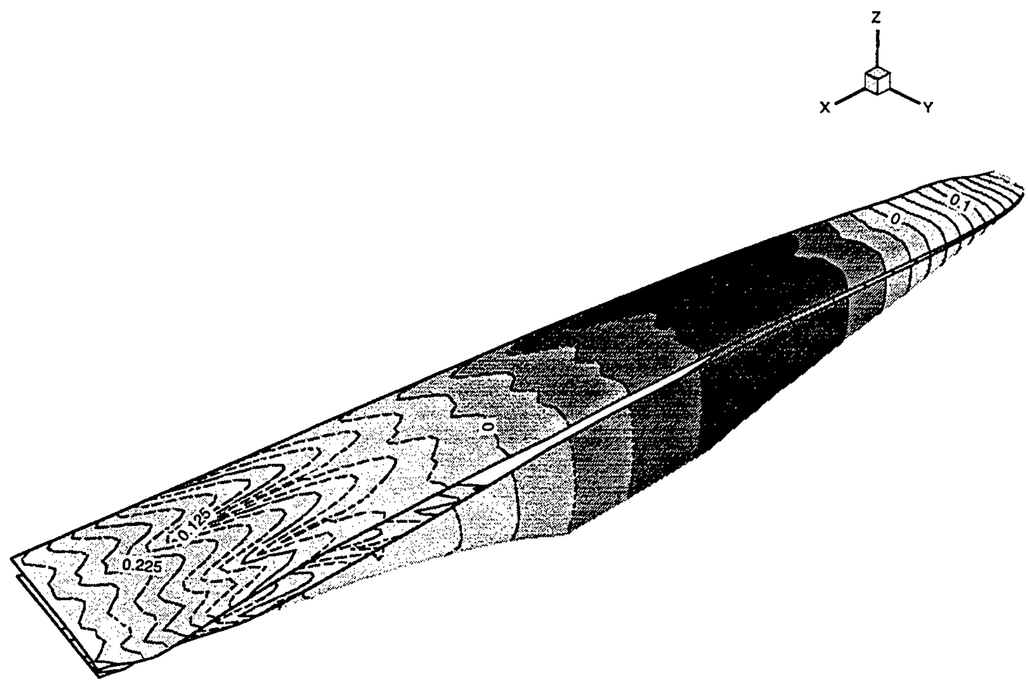


a) Fuselage Contours

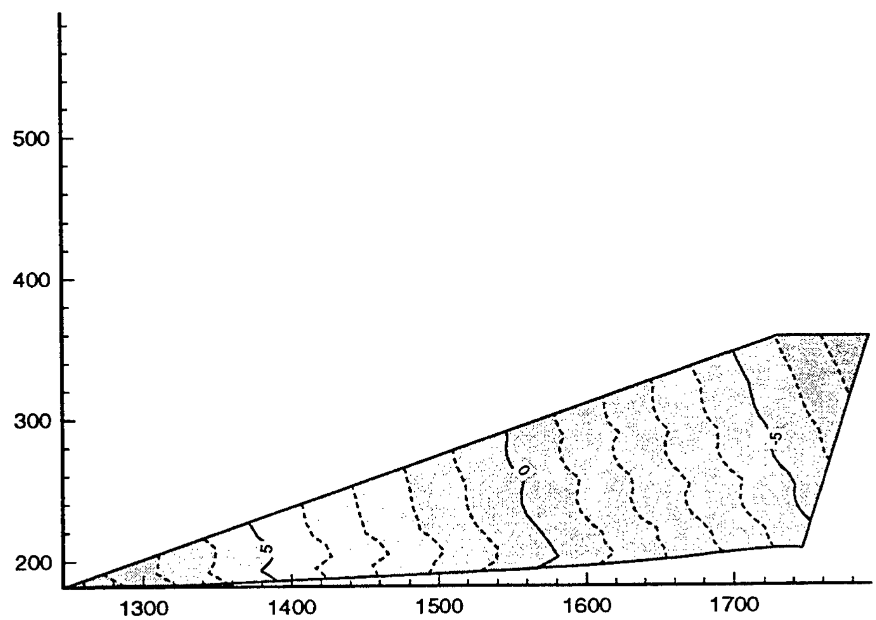


b) Wing Contours

Figure 13-41. Interpolation of Generic Hypersonic Model Mode 1 by Infinite-Plate Spline Method

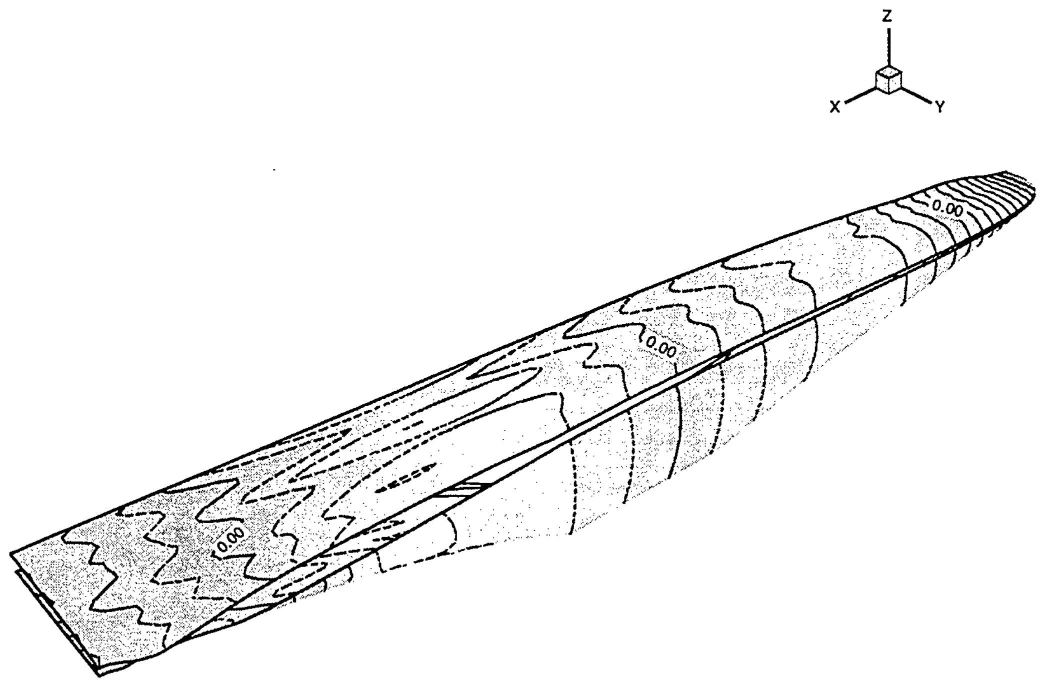


a) Fuselage Contours

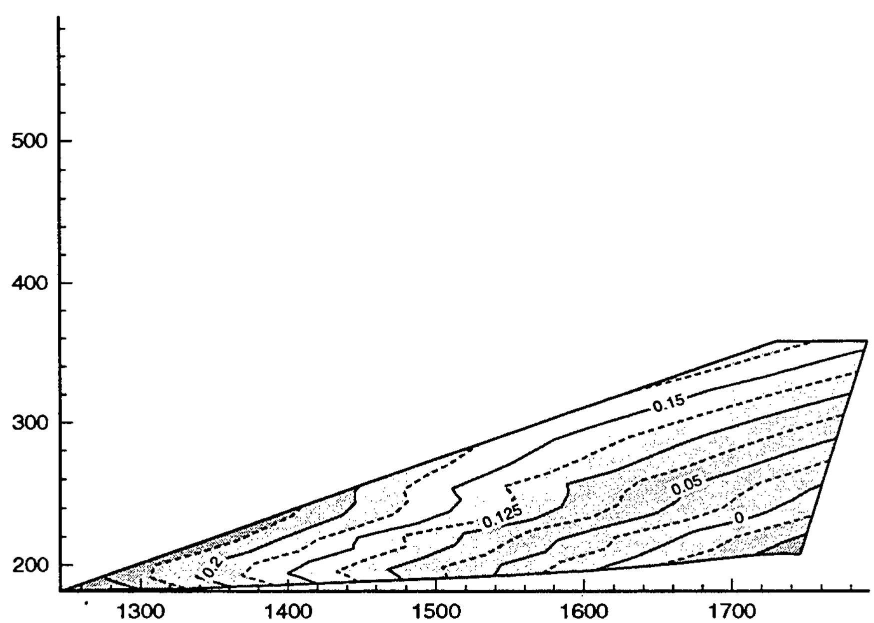


b) Wing Contours

Figure 13-42. Interpolation of Generic Hypersonic Model Mode 2 by Infinite-Plate Spline Method

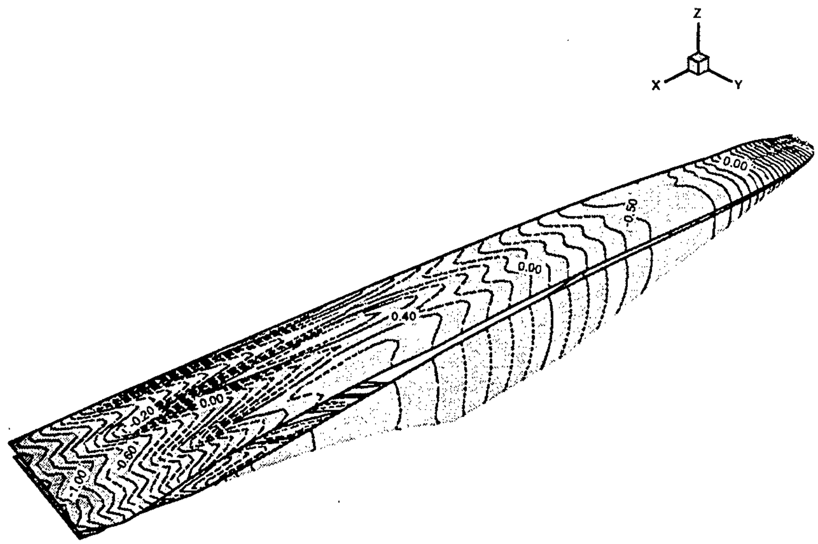


a) Fuselage Contours

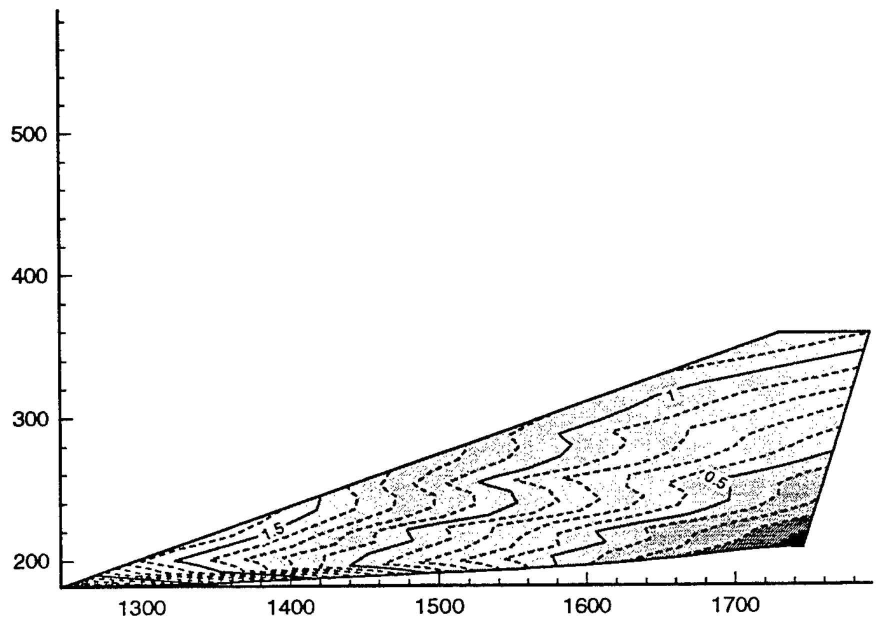


b) Wing Contours

Figure 13-43. Interpolation of Generic Hypersonic Model Mode 3 by Infinite-Plate Spline Method

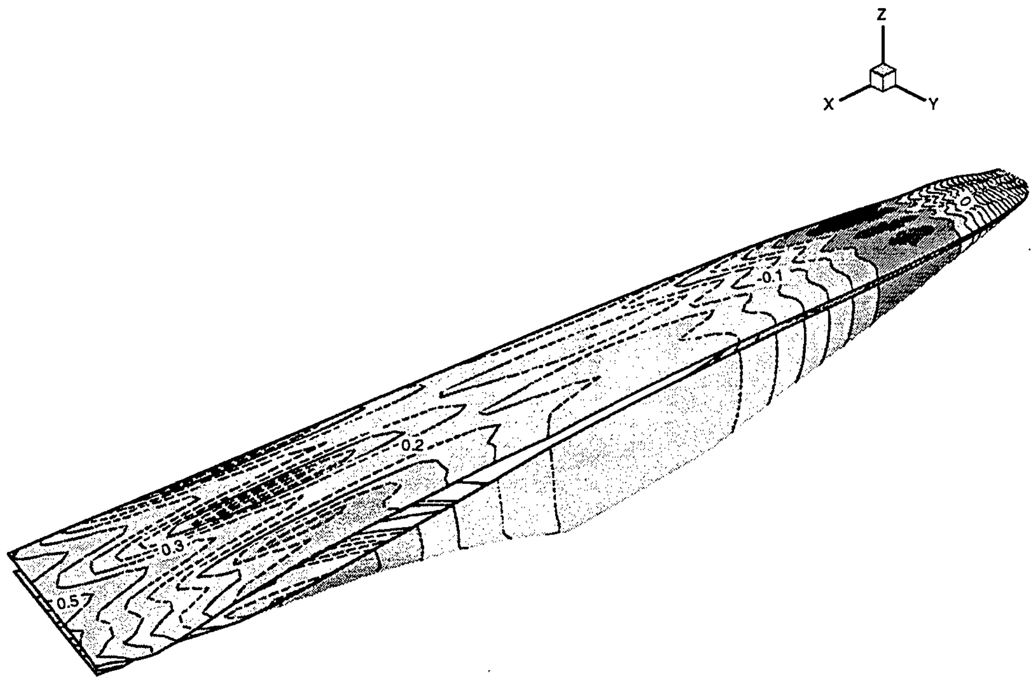


a) Fuselage Contours

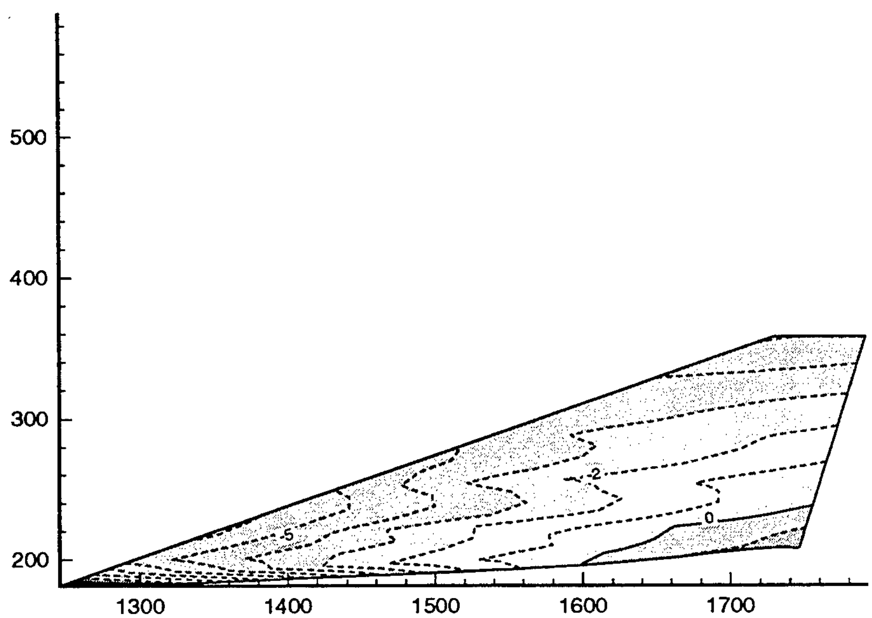


b) Wing Contours

Figure 13-44. Interpolation of Generic Hypersonic Model Mode 4 by Infinite-Plate Spline Method

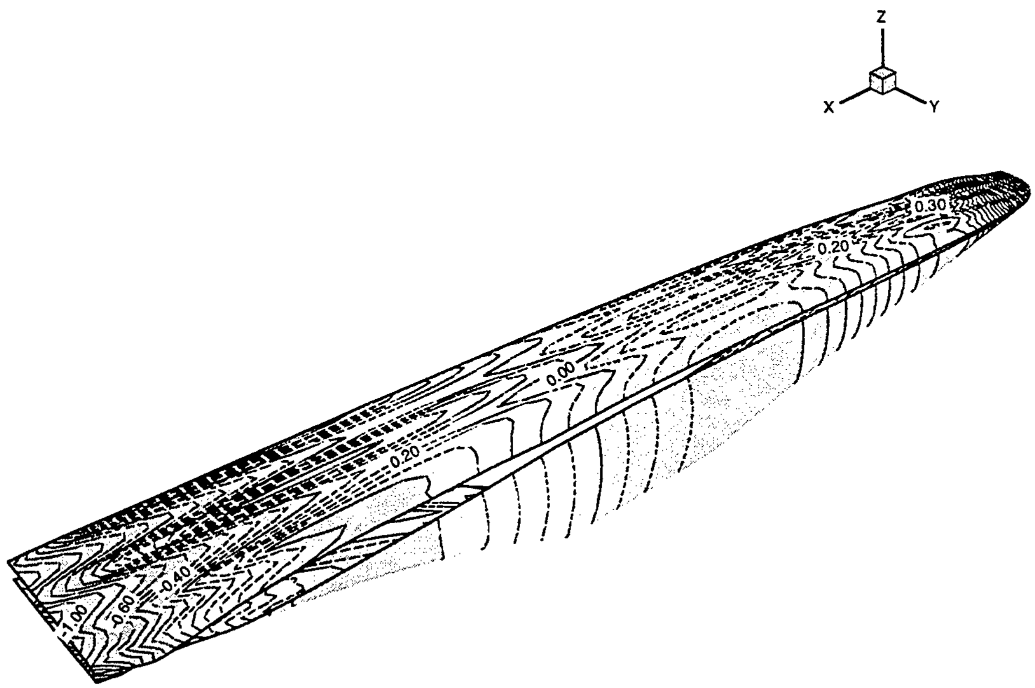


a) Fuselage Contours

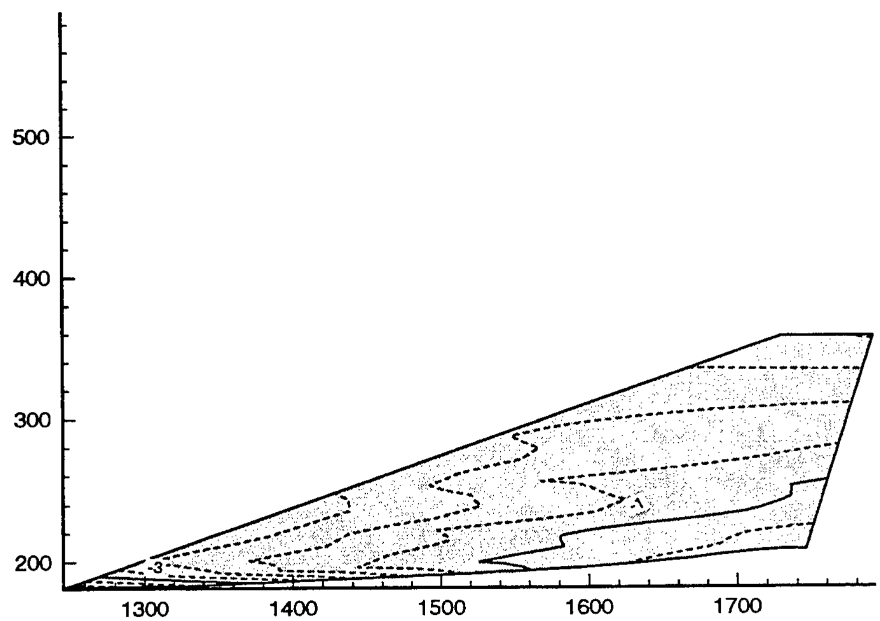


b) Wing Contours

Figure 13-45. Interpolation of Generic Hypersonic Model Mode 5 by Infinite-Plate Spline Method

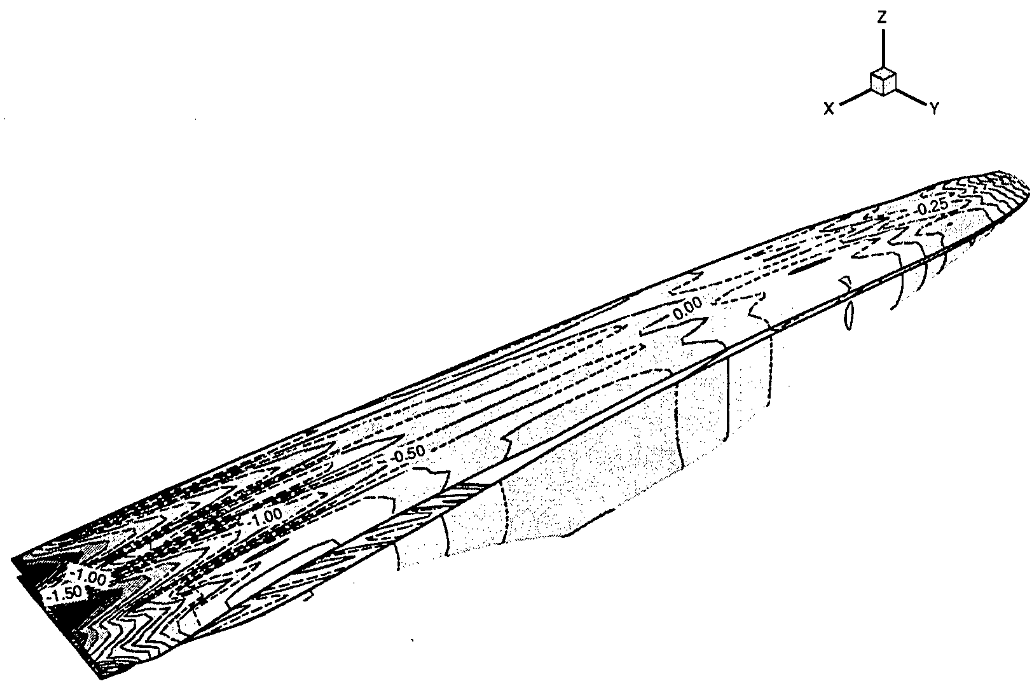


a) Fuselage Contours

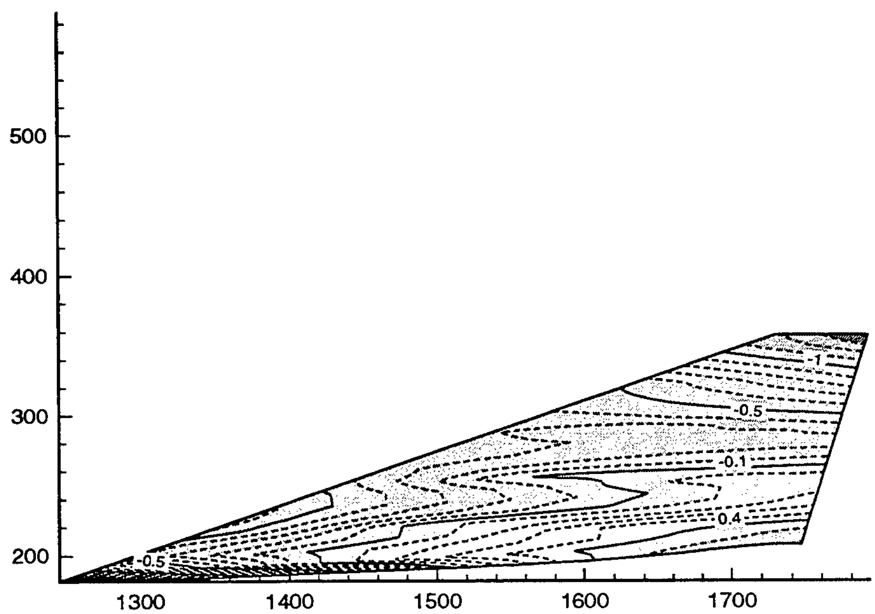


b) Wing Contours

Figure 13-46. Interpolation of Generic Hypersonic Model Mode 6 by Infinite-Plate Spline Method

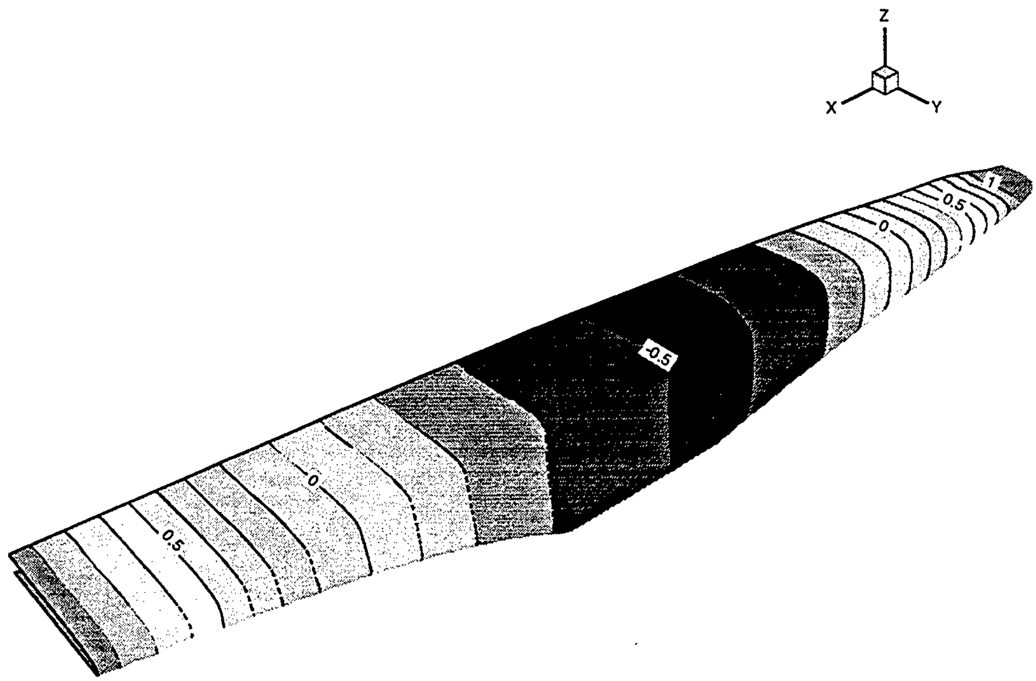


a) Fuselage Contours

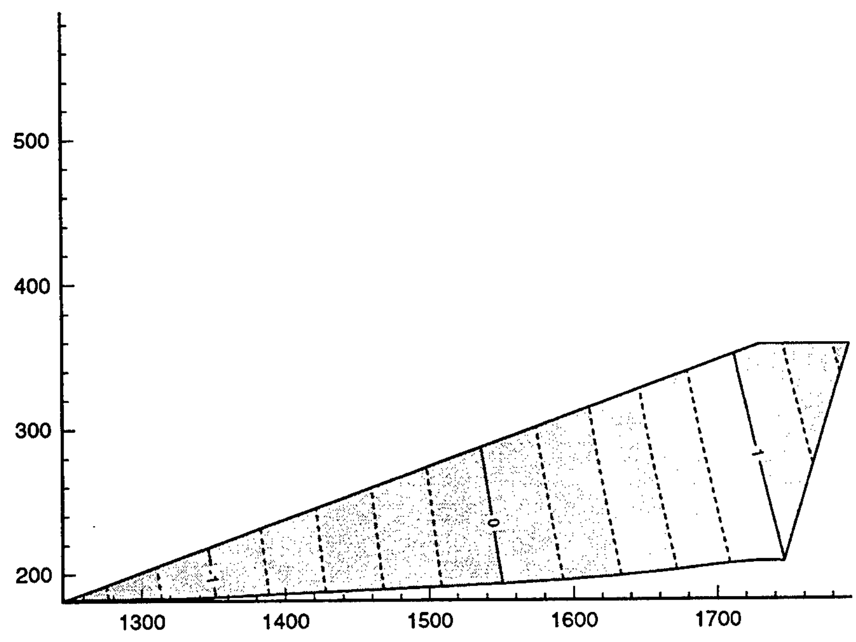


b) Wing Contours

Figure 13-47. Interpolation of Generic Hypersonic Model Mode 7 by Infinite-Plate Spline Method

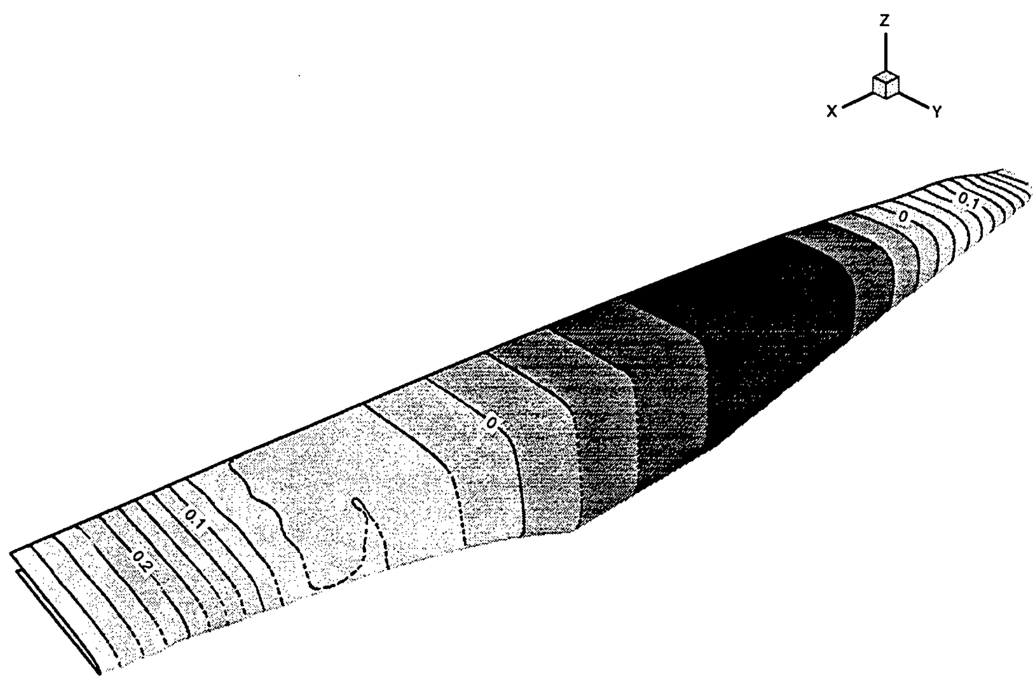


a) Fuselage Contours

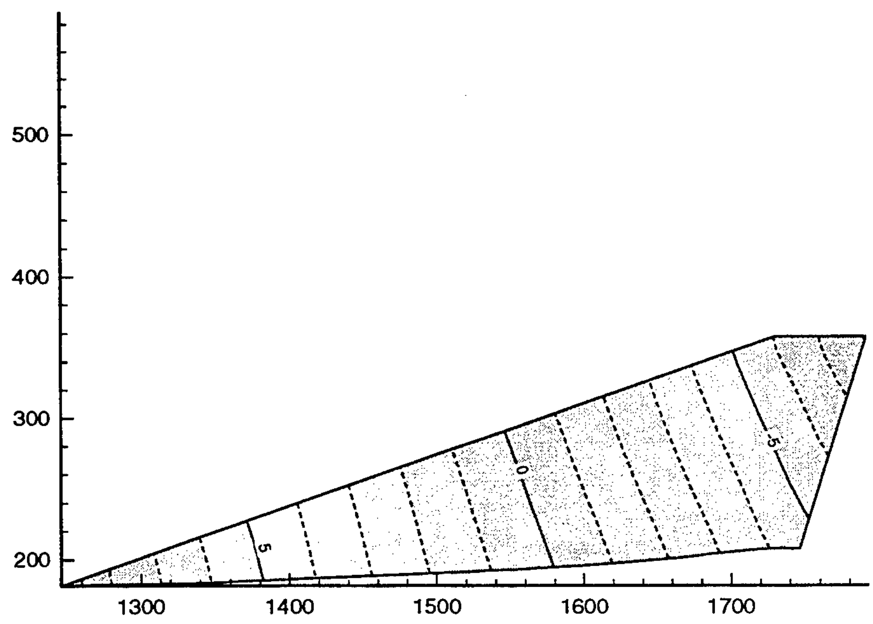


b) Wing Contours

Figure 13-48. Interpolation of Generic Hypersonic Model Mode Shape 1 by Multiquadrics with Scaling

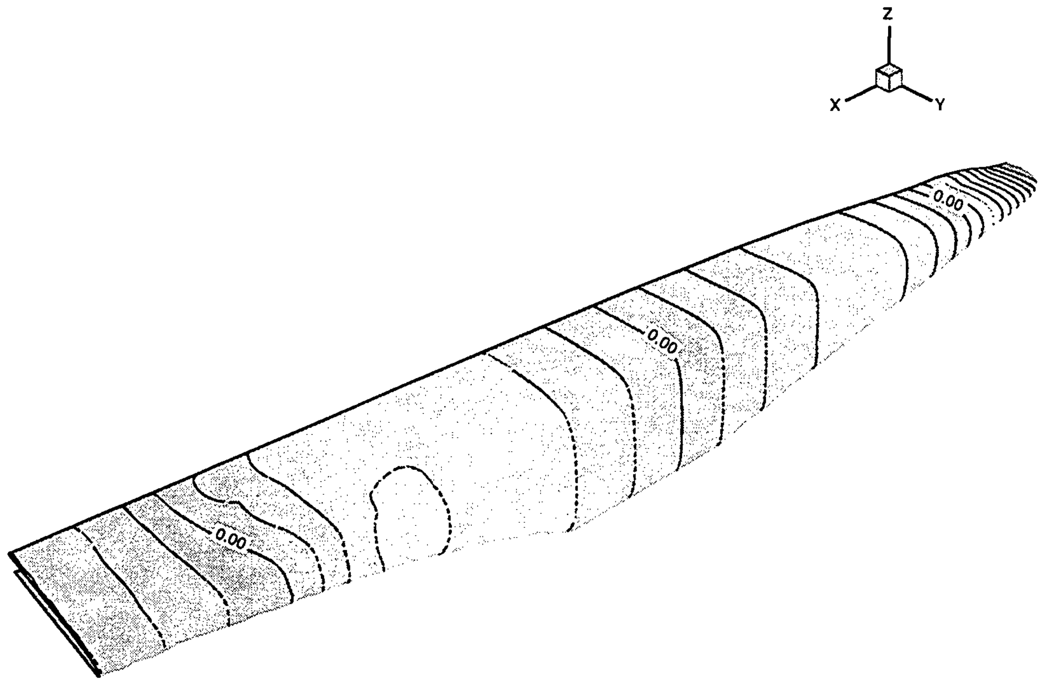


a) Fuselage Contours

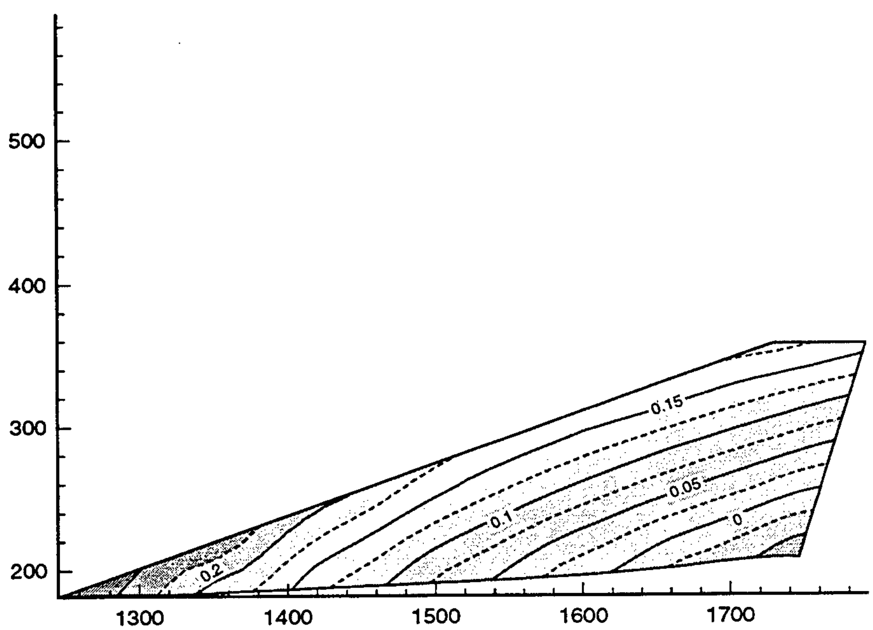


b) Wing Contours

Figure 13-49. Interpolation of Generic Hypersonic Model Mode Shape 2 by Multiquadrics with Scaling

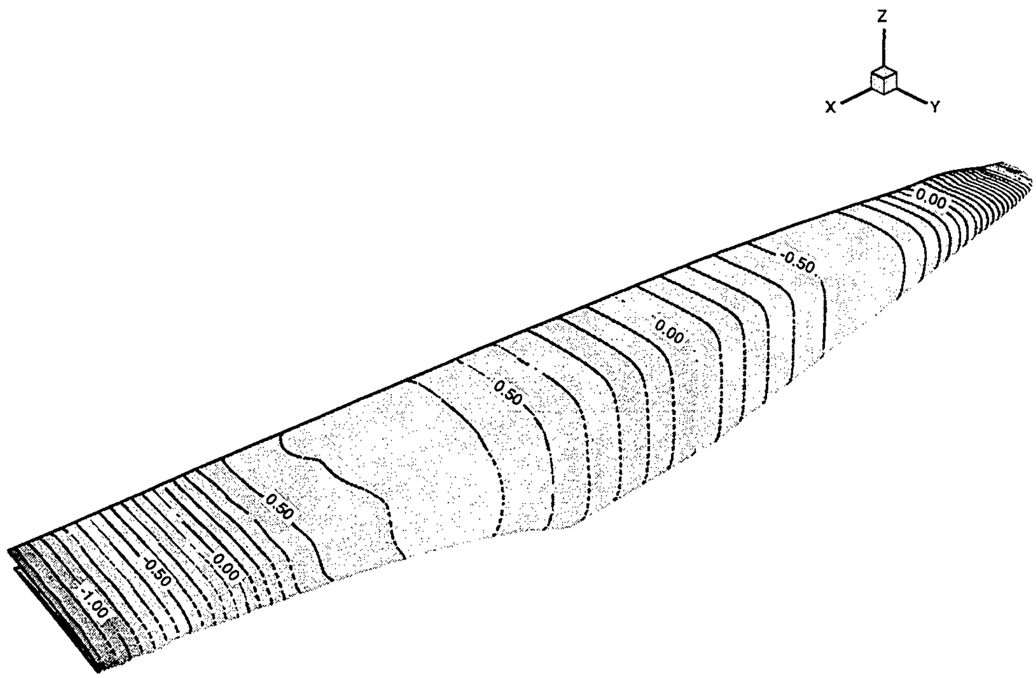


a) Fuselage Contours

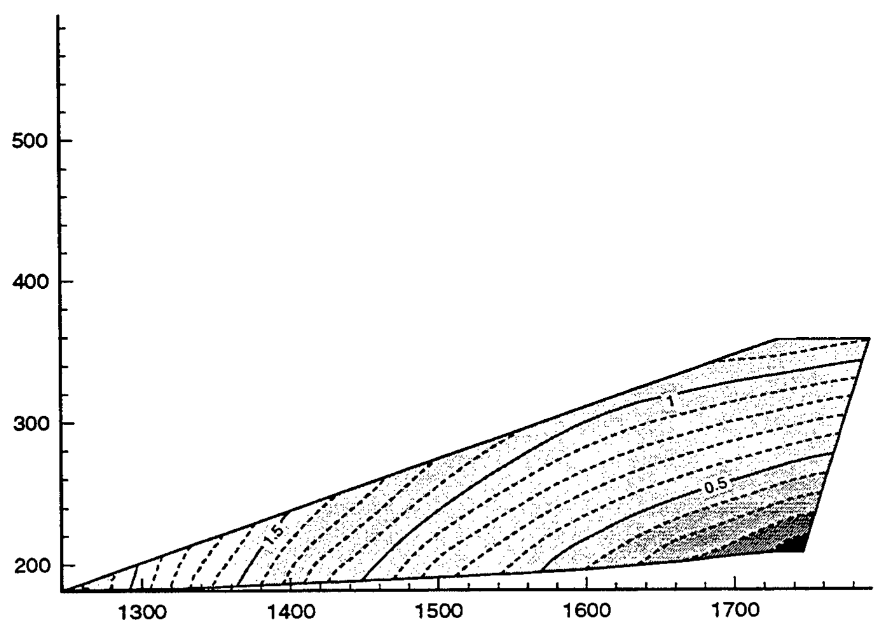


b) Wing Contours

Figure 13-50. Interpolation of Generic Hypersonic Model Mode Shape 3 by Multiquadrics with Scaling

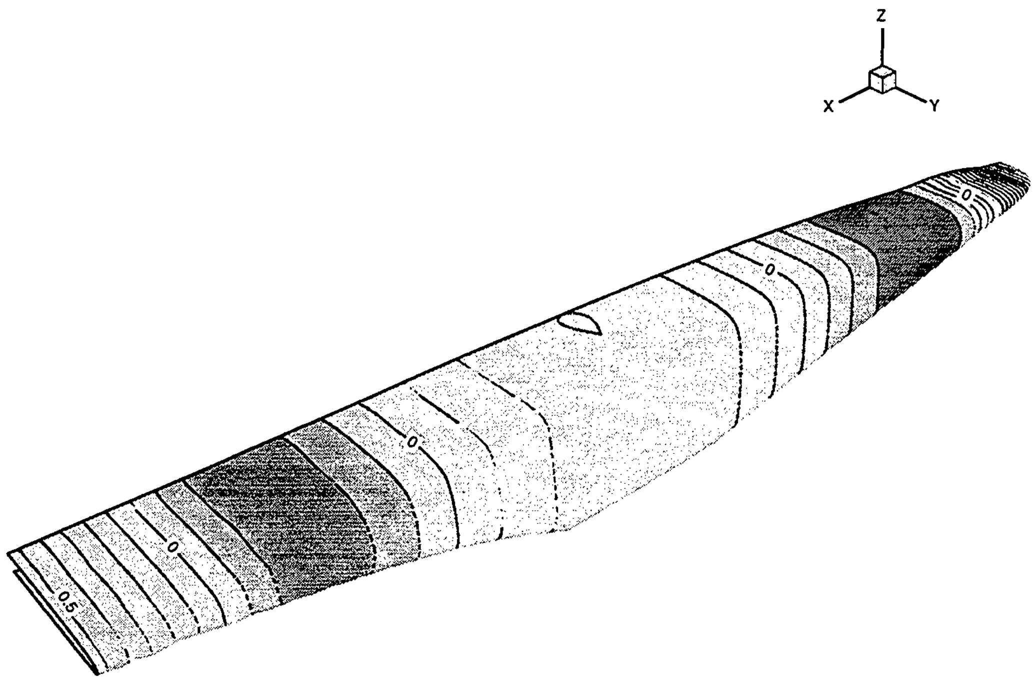


a) Fuselage Contours

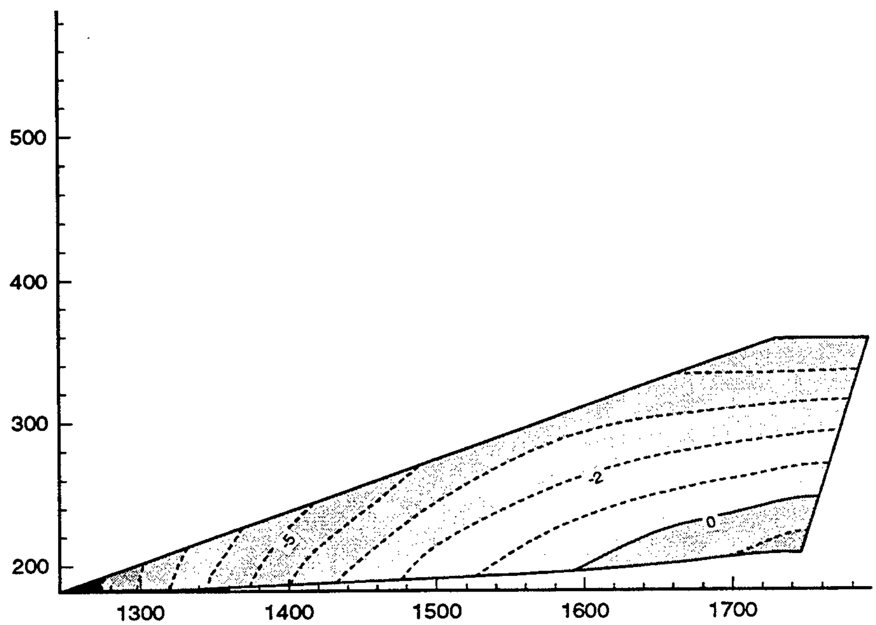


b) Wing Contours

Figure 13-51. Interpolation of Generic Hypersonic Model Mode Shape 4 by Multiquadrics with Scaling

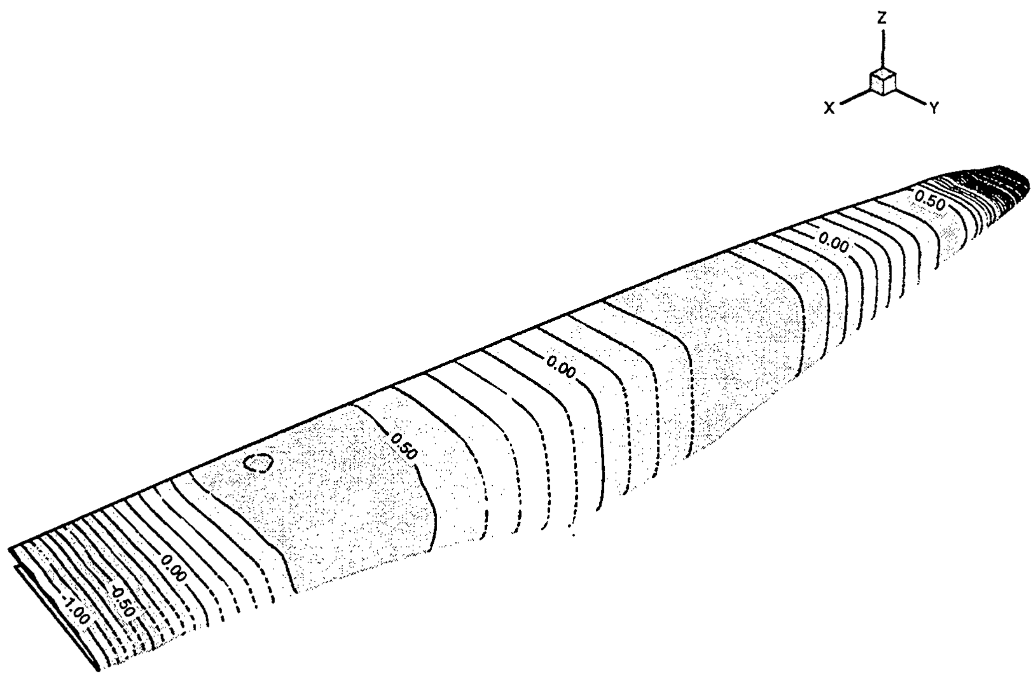


a) Fuselage Contours

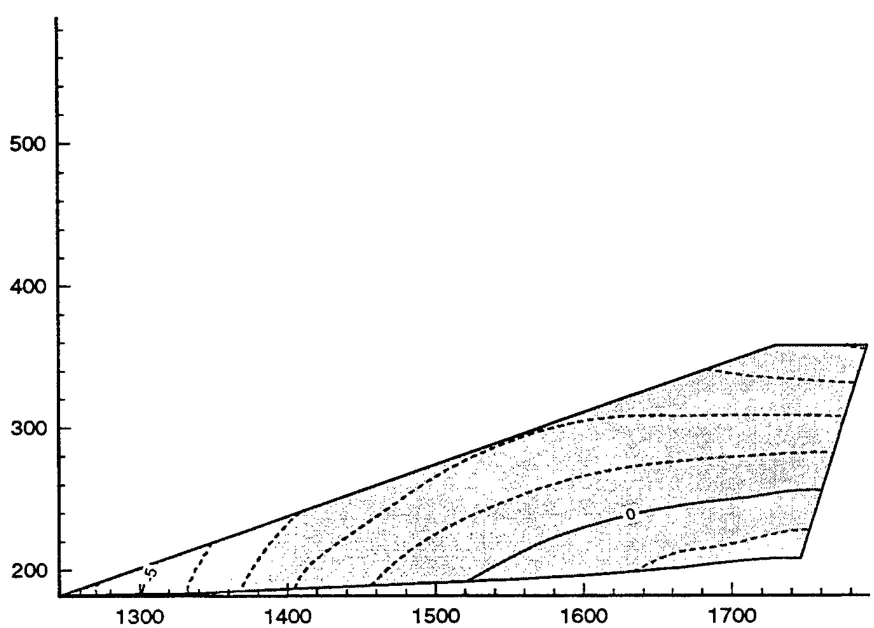


b) Wing Contours

Figure 13-52. Interpolation of Generic Hypersonic Model Mode Shape 5 by Multiquadrics with Scaling

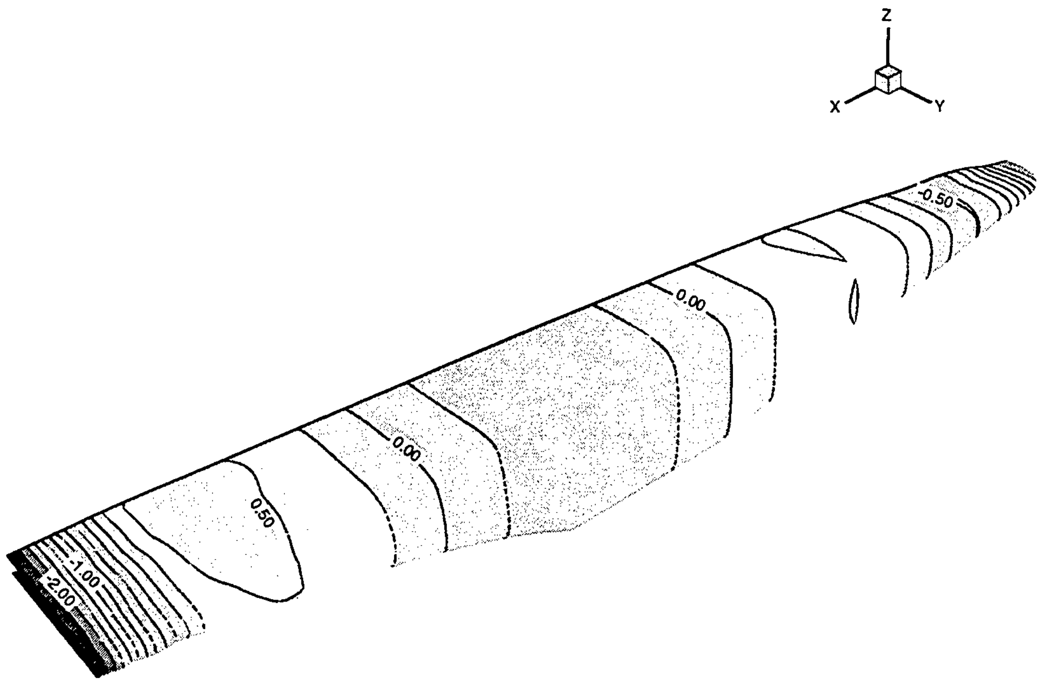


a) Fuselage Contours

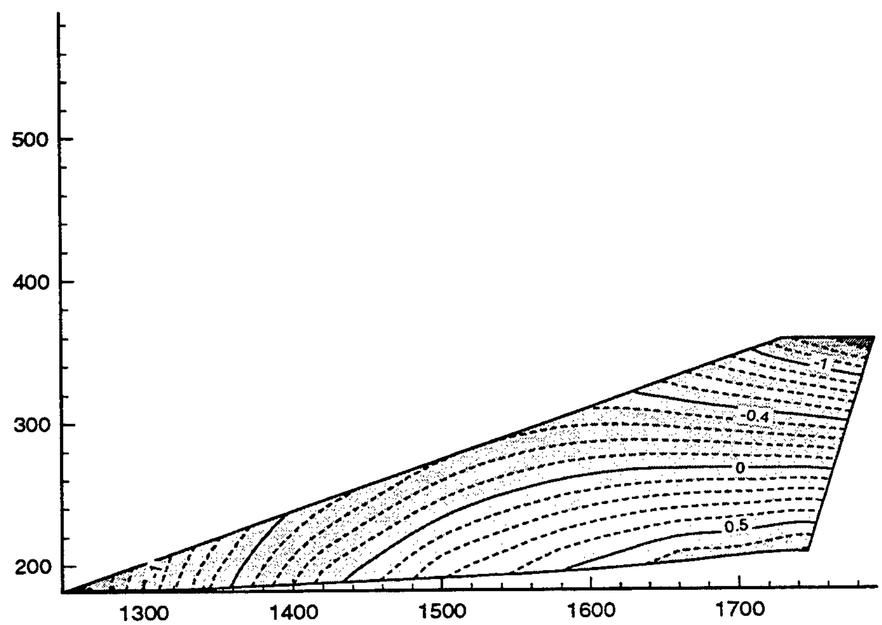


b) Wing Contours

Figure 13-53. Interpolation of Generic Hypersonic Model Mode Shape 6 by Multiquadrics with Scaling

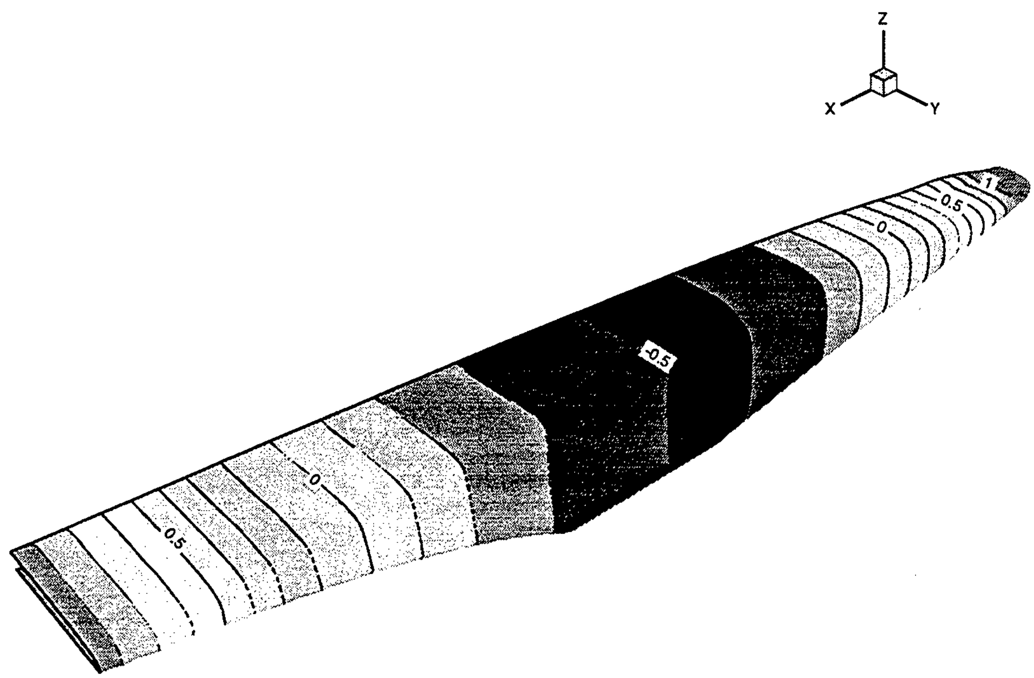


a) Fuselage Contours

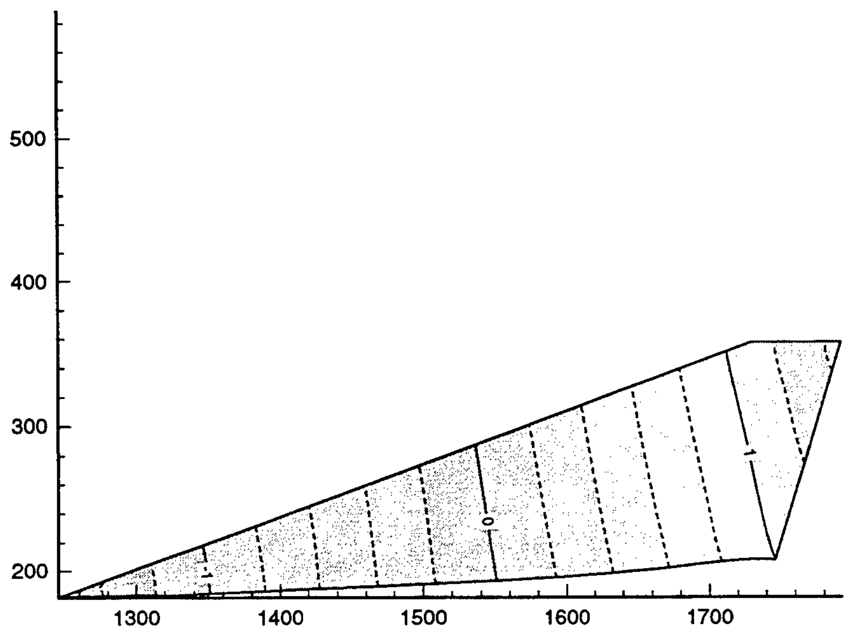


b) Wing Contours

Figure 13-54. Interpolation of Generic Hypersonic Model Mode Shape 7 by Multiquadrics with Scaling

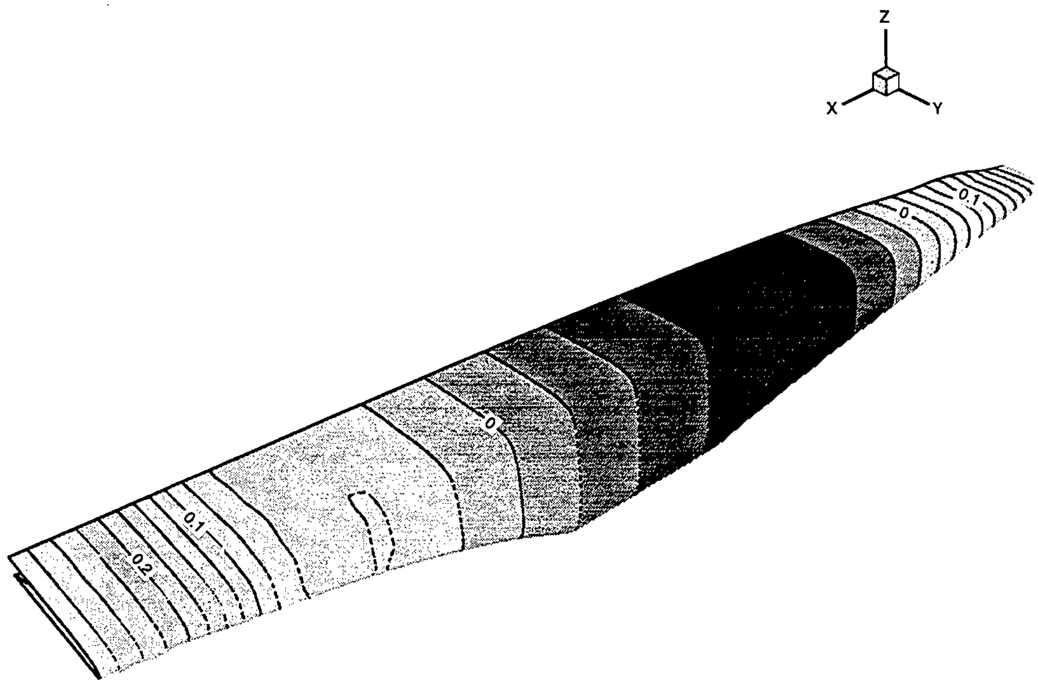


a) Fuselage Contours

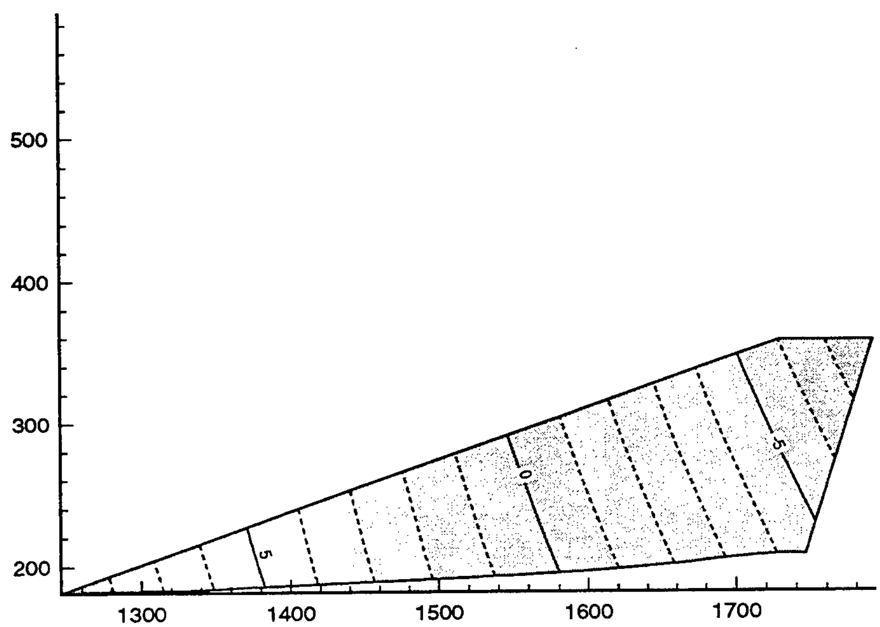


b) Wing Contours

Figure 13-55. Interpolation of Generic Hypersonic Model Mode Shape 1 by Thin-Plate Spline Method with Scaling

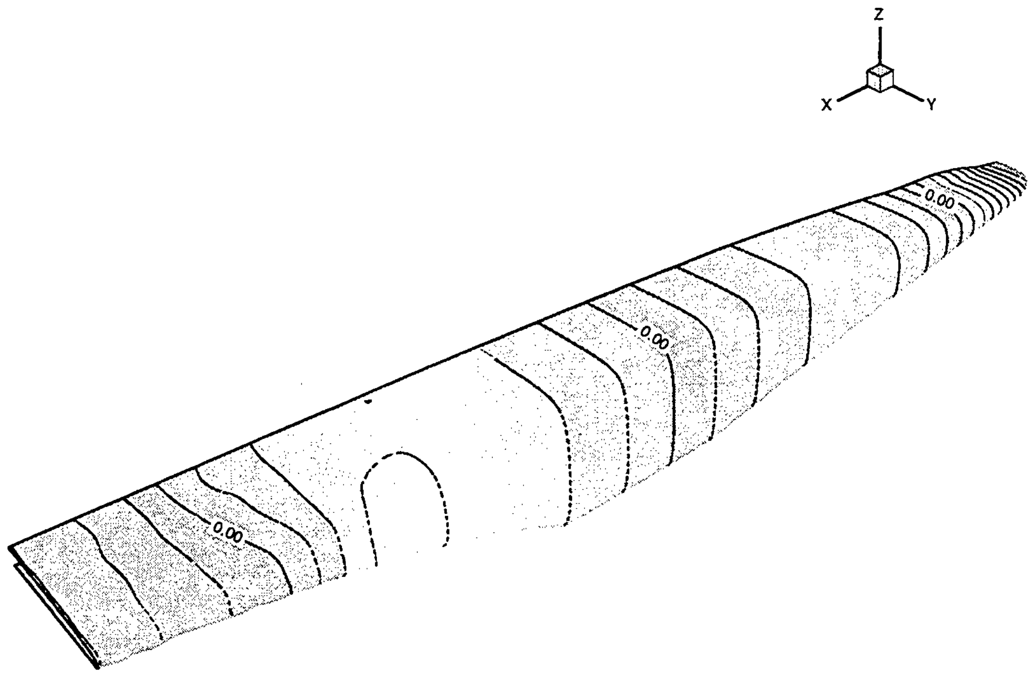


a) Fuselage Contours

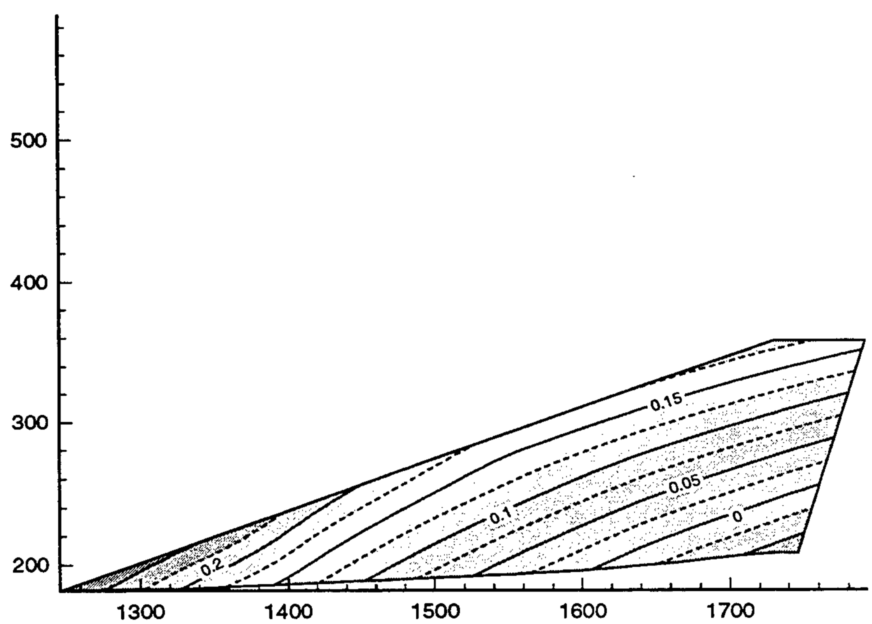


b) Wing Contours

Figure 13-56. Interpolation of Generic Hypersonic Model Mode Shape 2 by Thin-Plate Spline Method with Scaling

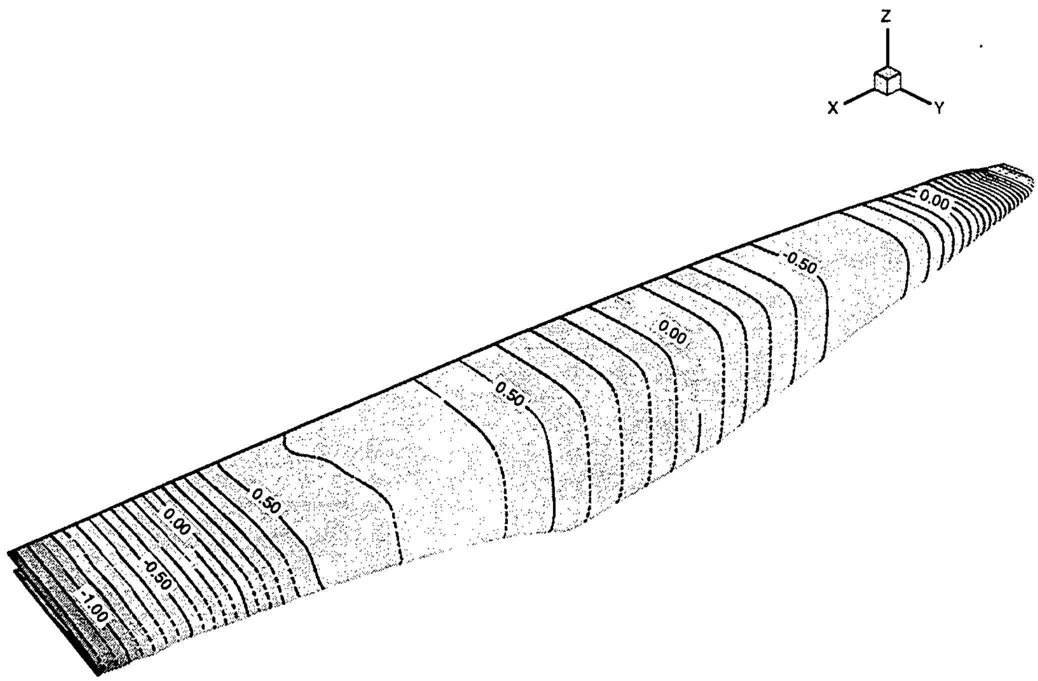


a) Fuselage Contours

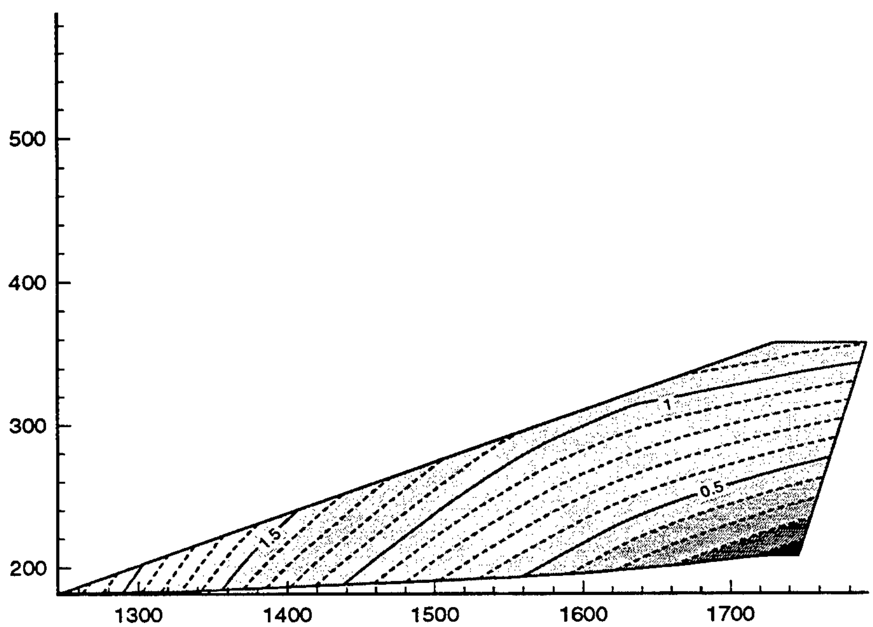


b) Wing Contours

Figure 13-57. Interpolation of Generic Hypersonic Model Mode Shape 3 by Thin-Plate Spline Method with Scaling

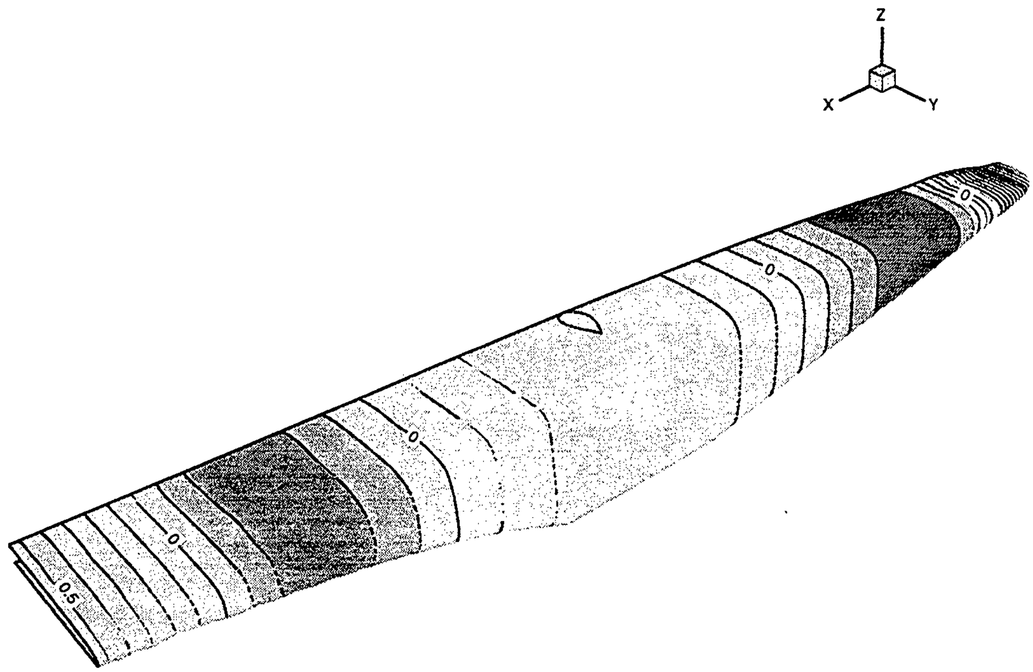


a) Fuselage Contours

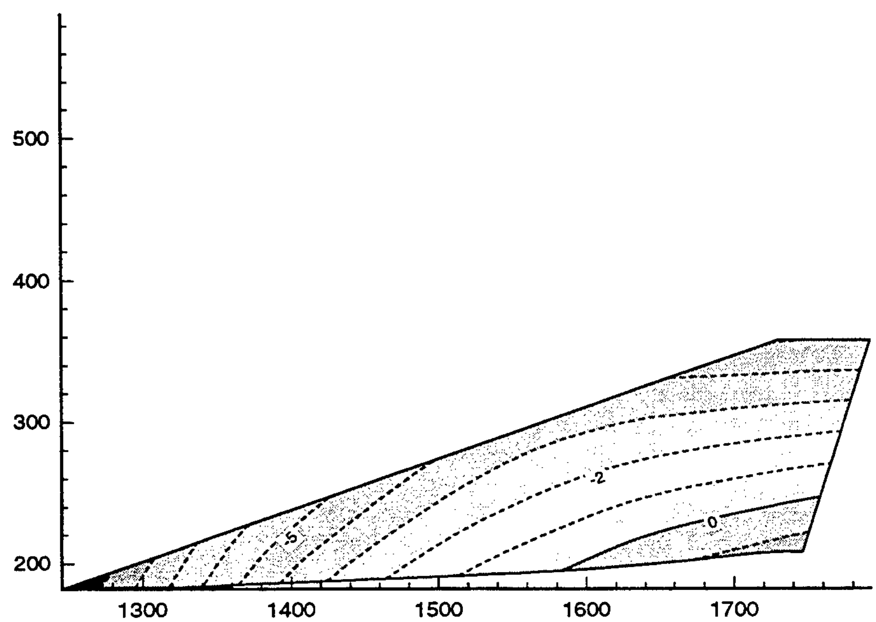


b) Wing Contours

Figure 13-58. Interpolation of Generic Hypersonic Model Mode Shape 4 by Thin-Plate Spline Method with Scaling

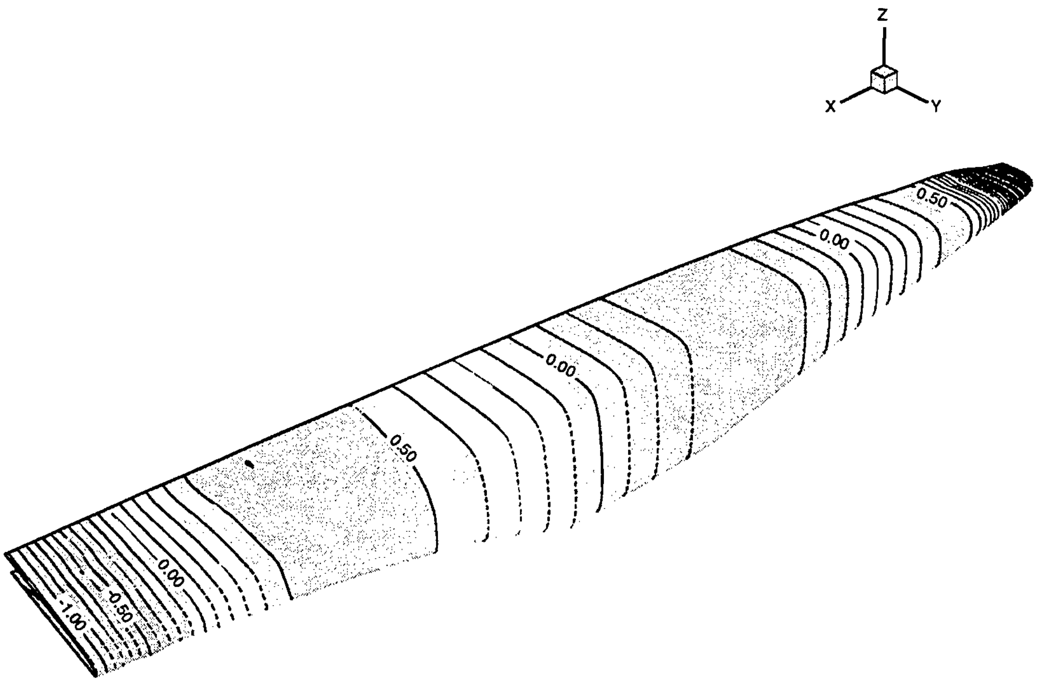


a) Fuselage Contours

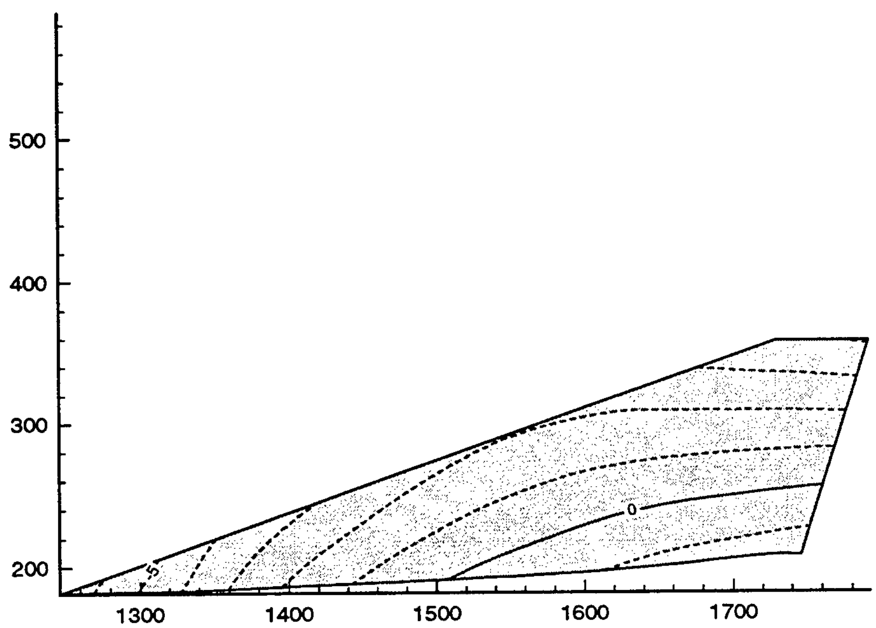


b) Wing Contours

Figure 13-59. Interpolation of Generic Hypersonic Model Mode Shape 5 by Thin-Plate Spline Method with Scaling

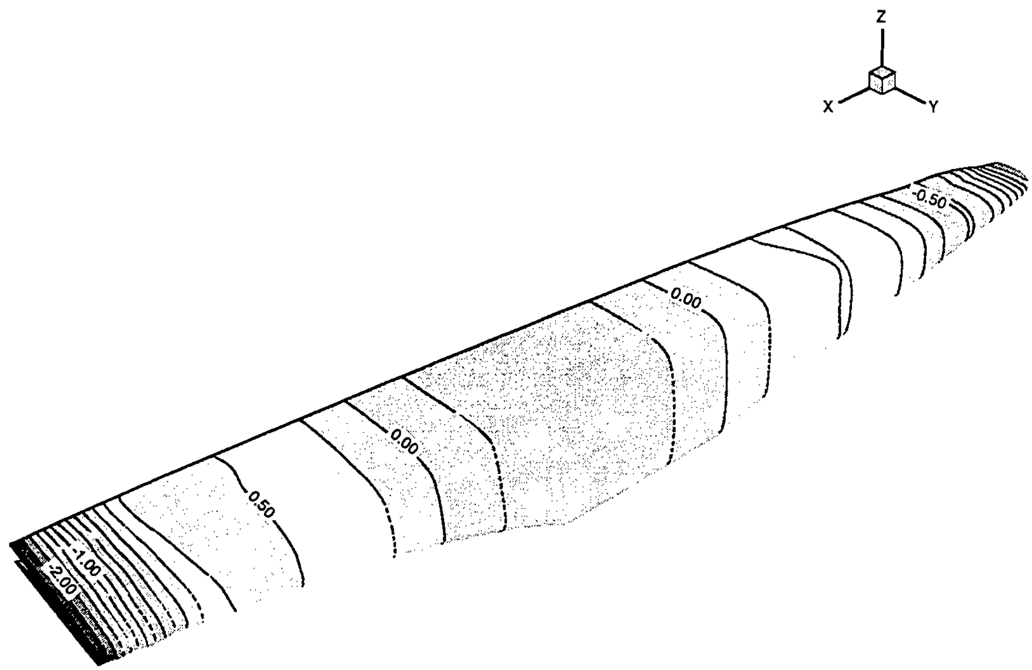


a) Fuselage Contours

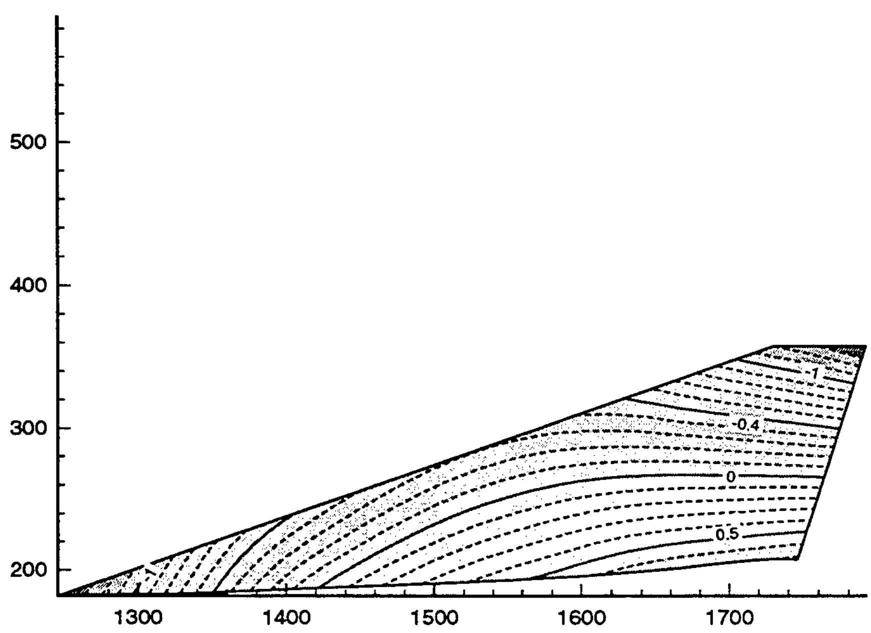


b) Wing Contours

Figure 13-60. Interpolation of Generic Hypersonic Model Mode Shape 6 by Thin-Plate Spline Method with Scaling



a) Fuselage Contours



b) Wing Contours

Figure 13-61. Interpolation of Generic Hypersonic Model Mode Shape 7 by Thin Plate Method with Scaling

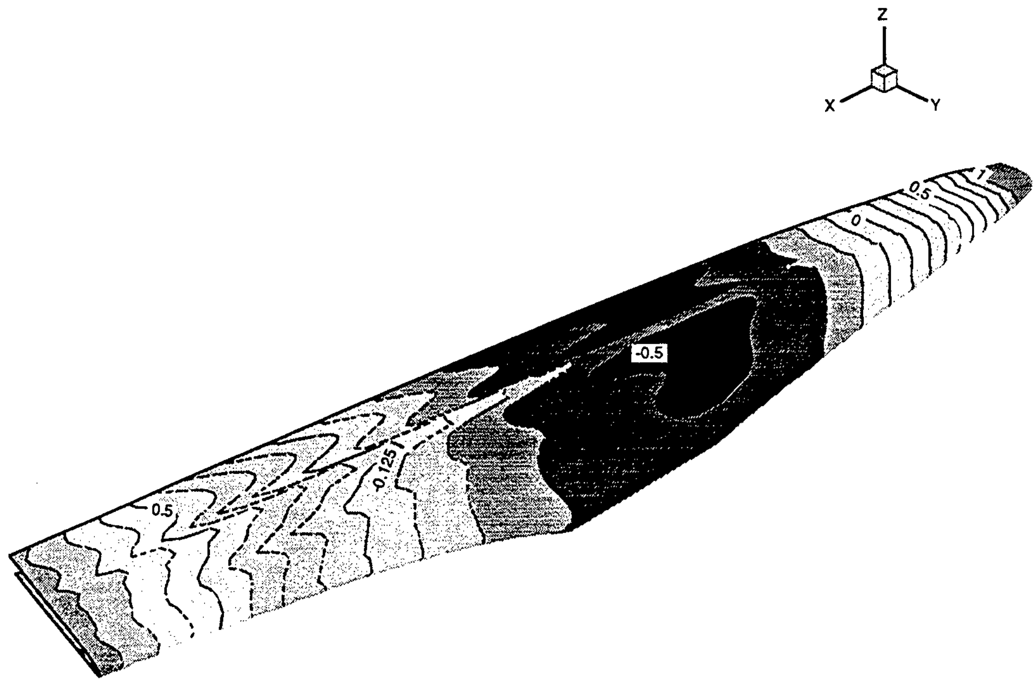


Figure 13-62. Interpolation of Generic Hypersonic Model Mode Shape 1 by Multiquadrics without Scaling

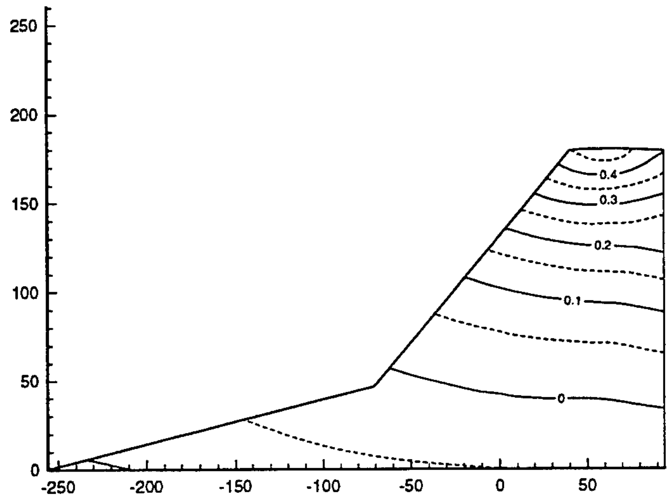
13.4 F-16 Wing and Strake

The IPS interpolation/extrapolation yields some interesting results, as shown in Figures 13-63 to 13-69. For modes 1 and 2, IPS does an excellent job of interpolation. At the higher modes where the motion of the control surface is evident, IPS has some minor problems. The interpolation of the modes outboard of the control surface are somewhat less than perfect. The concentration of the rapidly changing contours around the control surface also appears to shift the interpolation of the modes aft. This is particularly evident in modes 3, 4, 5 and 7. The extrapolation of the strake tends to be regular. The IPS method seems to extrapolate the contours linearly toward the root, based on the contours directly outboard. The control surface shape is continued to the root and is shifted forward. This forward shift is directly proportional to the overall chord or length of the strake.

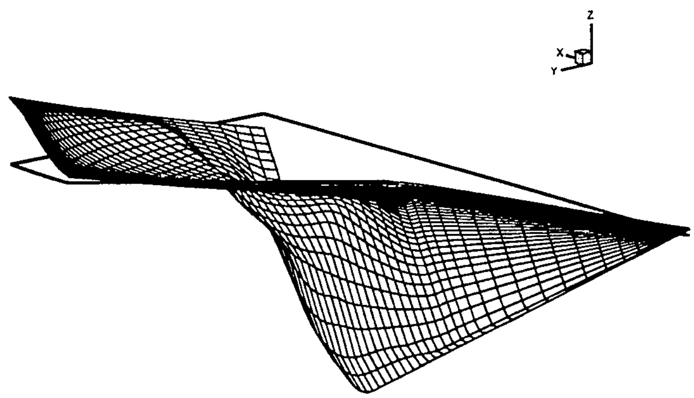
The MQ interpolation/extrapolation results are shown in Figures 13-70 – 13-76. For all of the modes, the MQ method does not reach the full amplitude of the mode. The MQ method has a tendency for this application to spread or distort the modes, as distinctly seen by comparing the contours. Because of this shift, large amplitudes near the edges are underpredicted, and the area over which they act is also reduced. Because this phenomenon did not occur for the previous, pure interpolation results, this spread could be the result of introducing the large extrapolation area of the strake. Extrapolation onto the strake takes two forms, depending on the orientation of the mode. For modes which end almost perpendicular to the structural wing root, the mode contour is extended to the strake root, and is shifted slightly forward. This forward shift is much smaller than that observed when using IPS. For mode contours which lie in a near-parallel orientation to the root, MQ does not extend the mode onto the strake. Thus, the zero deflections noted in modes 1, 2 and 3 at the wing root are extrapolated as zero or near-zero deflections onto the strake.

The TPS interpolation/extrapolation results are shown in Figures 13-77 – 13-83. It is evident by comparing these results with the MQ and IPS results that the TPS results are a combination of the MQ and IPS results. That is, the extrapolation characteristics seen in the IPS results are almost duplicated here. In addition, the spreading or shift of the mode contours exhibited by MQ is also reproduced here.

Thus, it appears that, for reproduction of the mode shapes, the IPS method does a much more accurate interpolation that does MQ and TPS on the given wing surface. However, if the strake should have zero or near-zero deflections, MQ does the best extrapolation onto the strake.

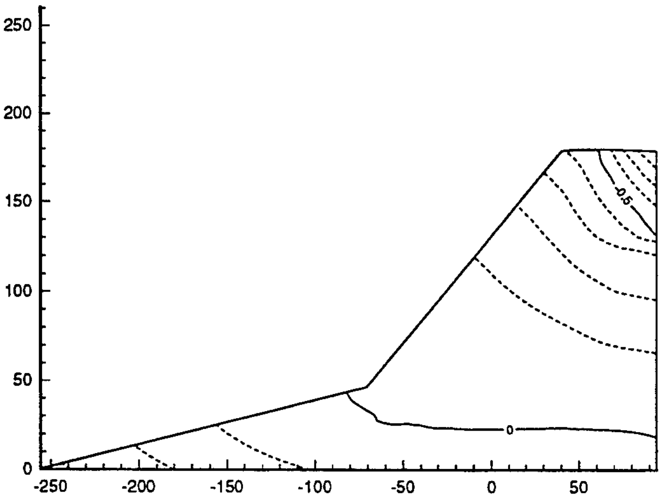


a) Mode Shape Contours

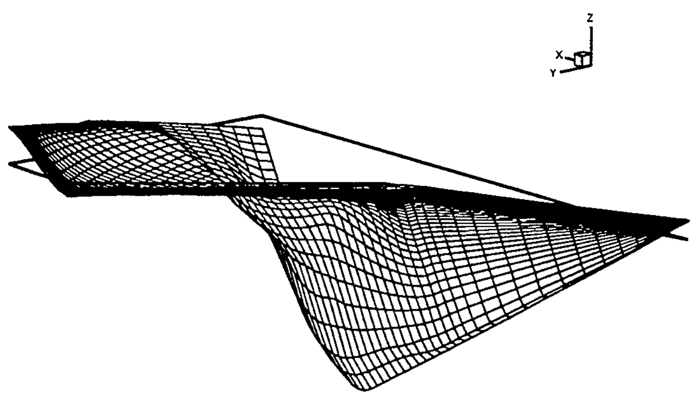


b) Mode Shape Deflections

Figure 13-63. F-16 Wing and Strake Mode Shape 1 Interpolations Using the Infinite Plate Spline Method

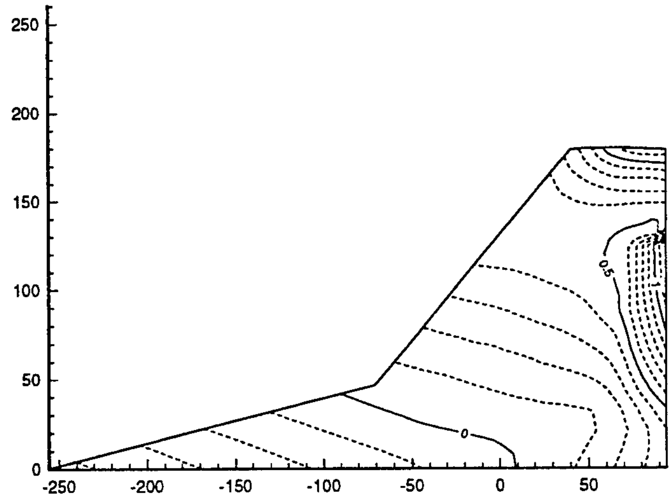


a) Mode Shape Contours

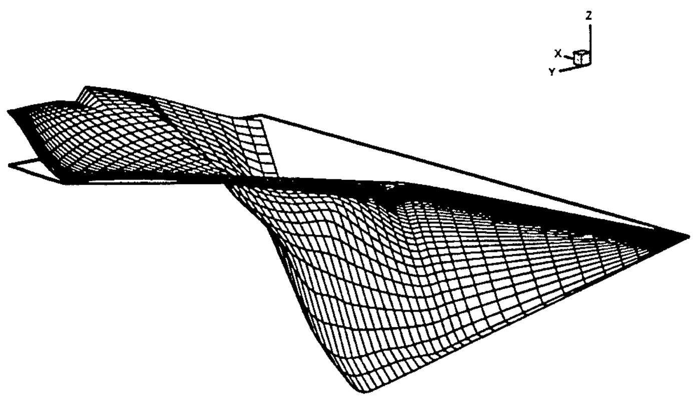


b) Mode Shape Deflections

Figure 13-64. F-16 Wing and Strake Mode Shape 2 Interpolations Using the Infinite Plate Spline Method

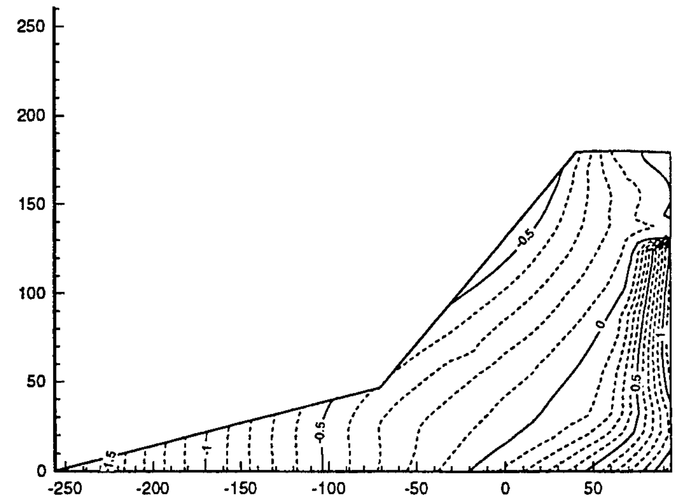


a) Mode Shape Contours

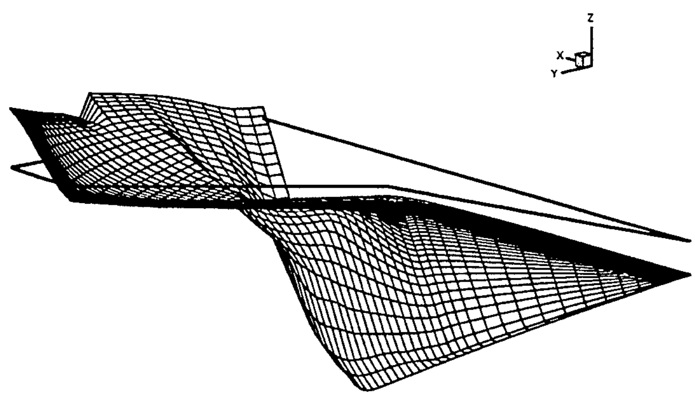


b) Mode Shape Deflections

Figure 13-65. F-16 Wing and Strake Mode Shape 3 Interpolations Using the Infinite Plate Spline Method

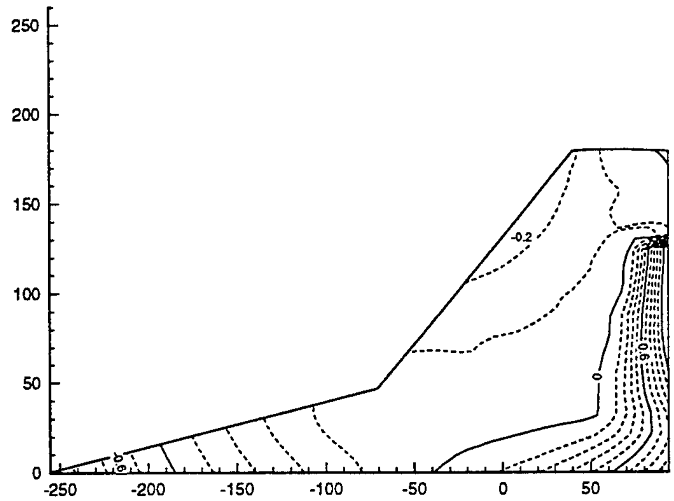


a) Mode Shape Contours

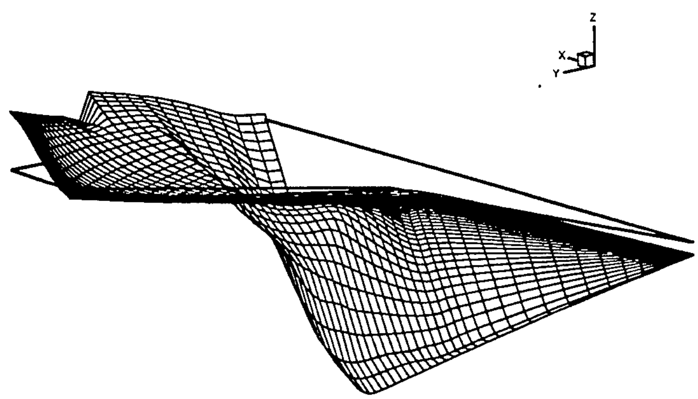


b) Mode Shape Deflections

Figure 13-66. F-16 Wing and Strake Mode Shape 4 Interpolations Using the Infinite Plate Spline Method

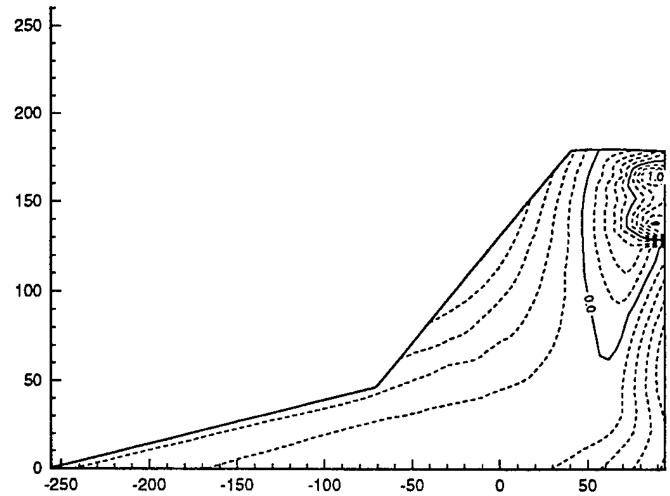


a) Mode Shape Contours

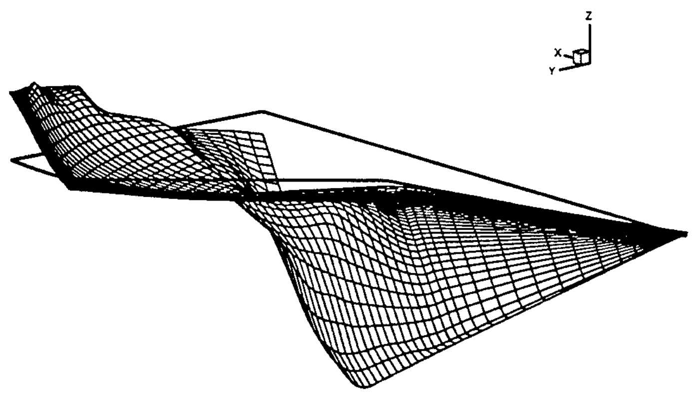


b) Mode Shape Deflections

Figure 13-67. F-16 Wing and Strake Mode Shape 5 Interpolations Using the Infinite Plate Spline Method

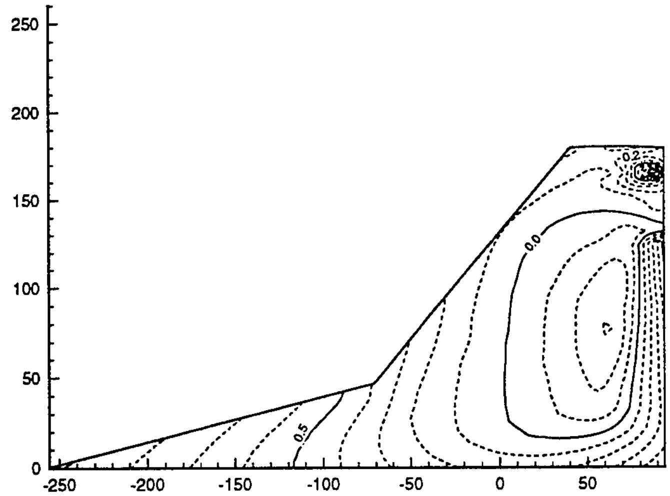


a) Mode Shape Contours

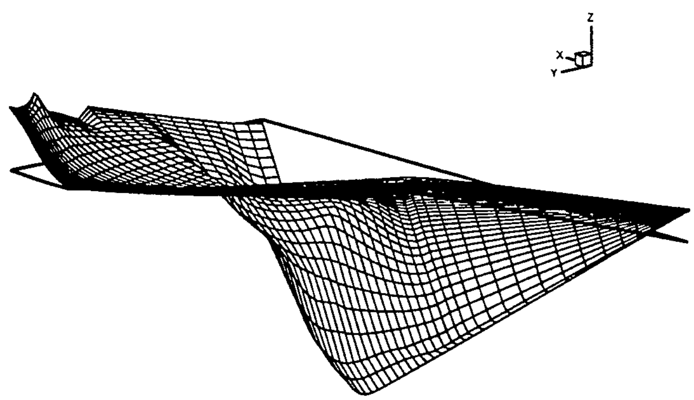


b) Mode Shape Deflections

Figure 13-68. F-16 Wing and Strake Mode Shape 6 Interpolations Using the Infinite Plate Spline Method

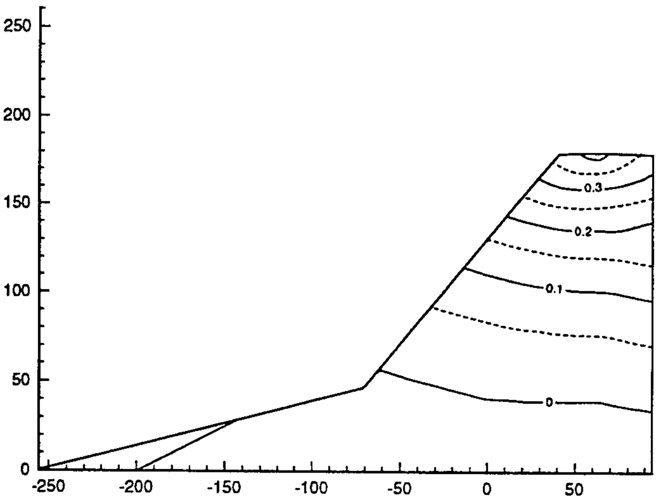


a) Mode Shape Contours

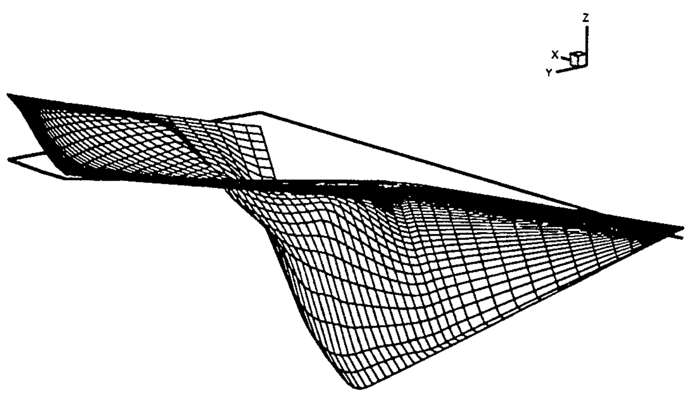


b) Mode Shape Deflections

Figure 13-69. F-16 Wing and Strake Mode Shape 7 Interpolations Using the Infinite Plate Spline Method

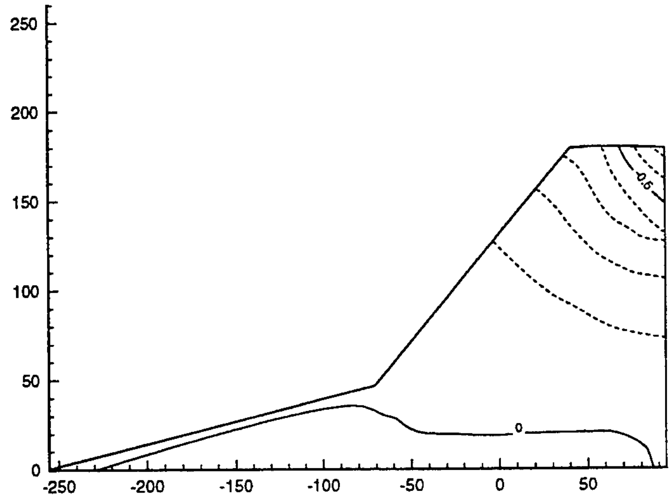


a) Mode Shape Contours

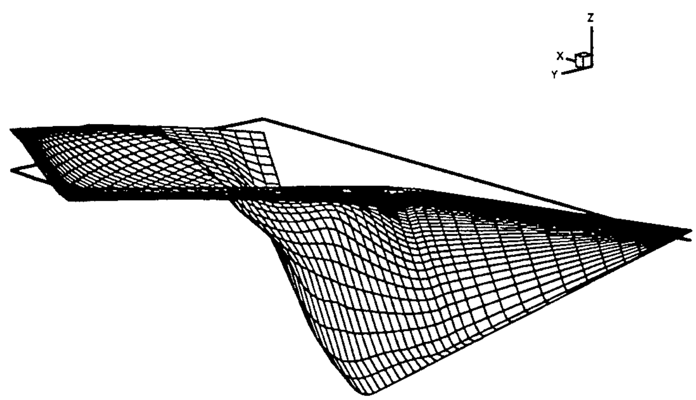


b) Mode Shape Deflections

Figure 13-70. F-16 Wing and Strake Mode Shape 1 Interpolations Using the Multiquadrics Method

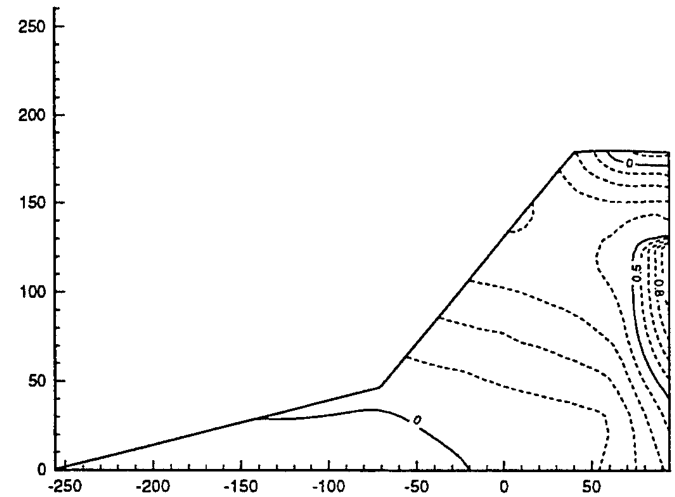


a) Mode Shape Contours

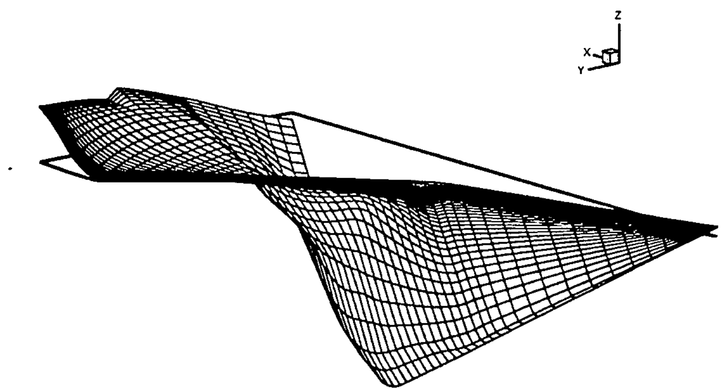


b) Mode Shape Deflections

Figure 13-71. F-16 Wing and Strake Mode Shape 2 Interpolations Using the Multiquadrics Method

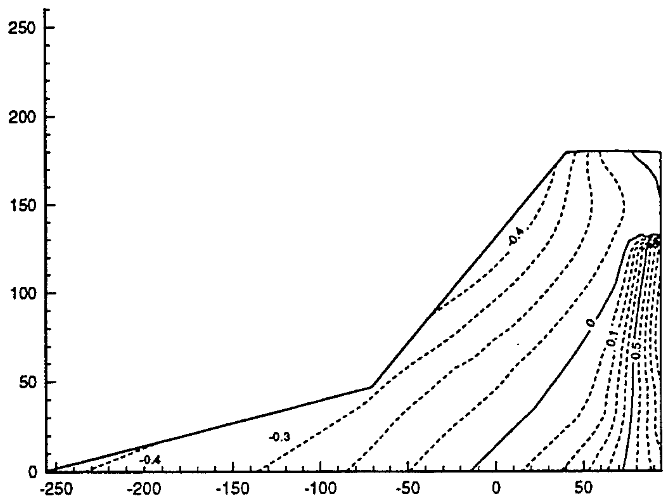


a) Mode Shape Contours

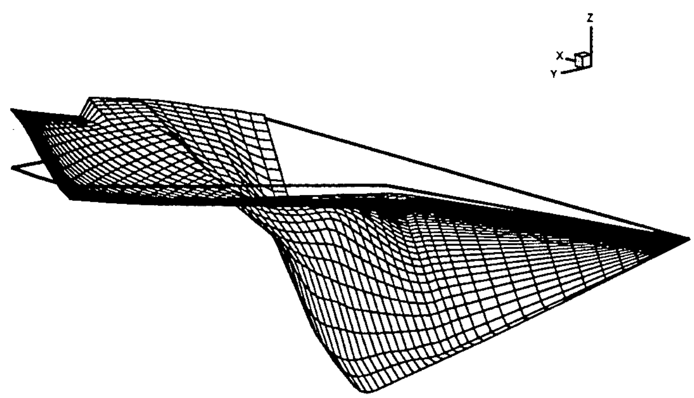


b) Mode Shape Deflections

Figure 13-72. F-16 Wing and Strake Mode Shape 3 Interpolations Using the Multiquadrics Method

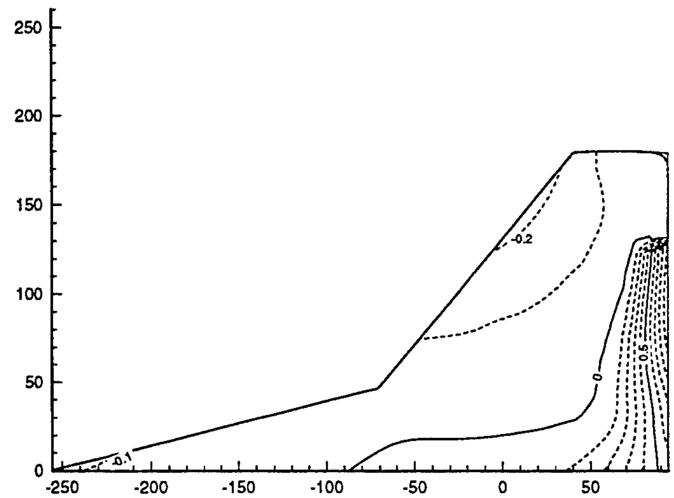


a) Mode Shape Contours

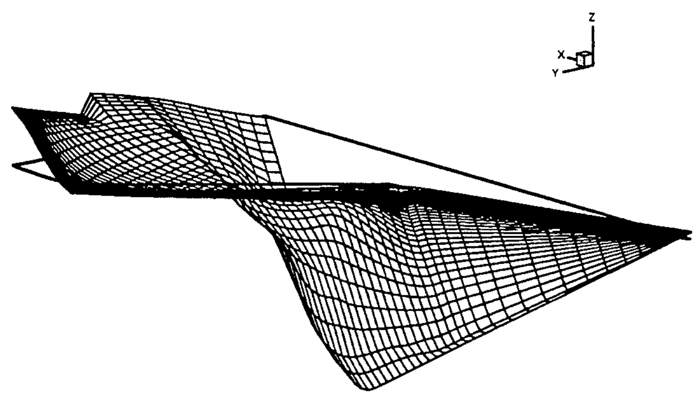


b) Mode Shape Deflections

Figure 13-73. F-16 Wing and Strake Mode Shape 4 Interpolations Using the Multiquadrics Method

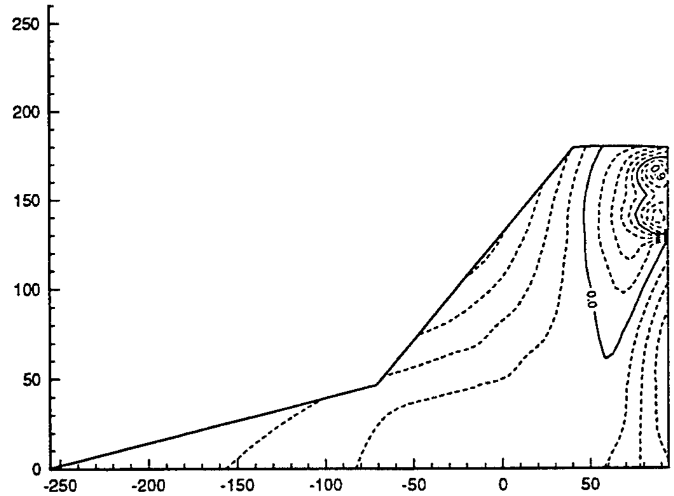


a) Mode Shape Contours

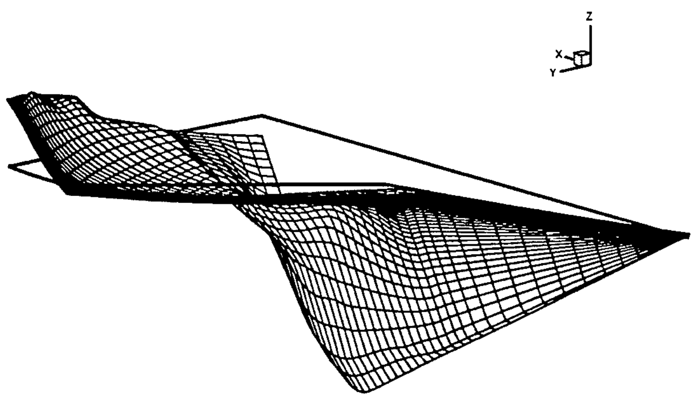


b) Mode Shape Deflections

Figure 13-74. F-16 Wing and Strake Mode Shape 5 Interpolations Using the Multiquadrics Method

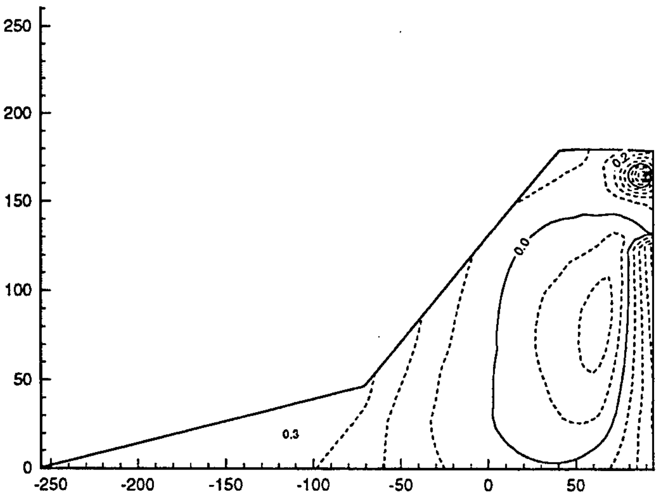


a) Mode Shape Contours

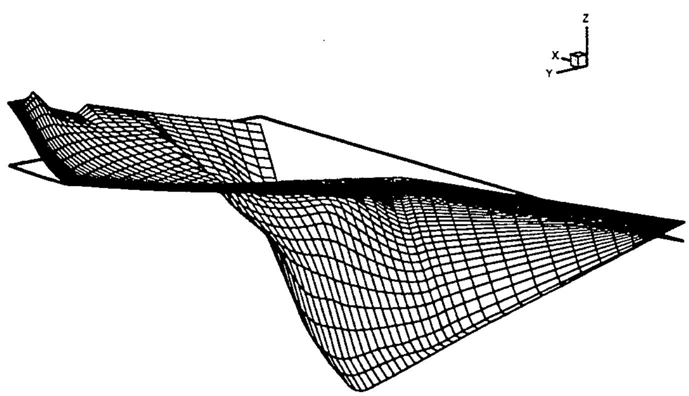


b) Mode Shape Deflections

Figure 13-75. F-16 Wing and Strake Mode Shape 6 Interpolations Using the Multiquadrics Method

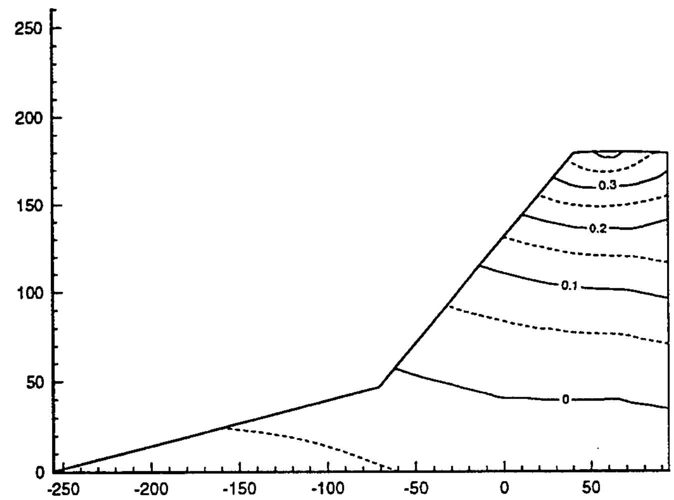


a) Mode Shape Contours

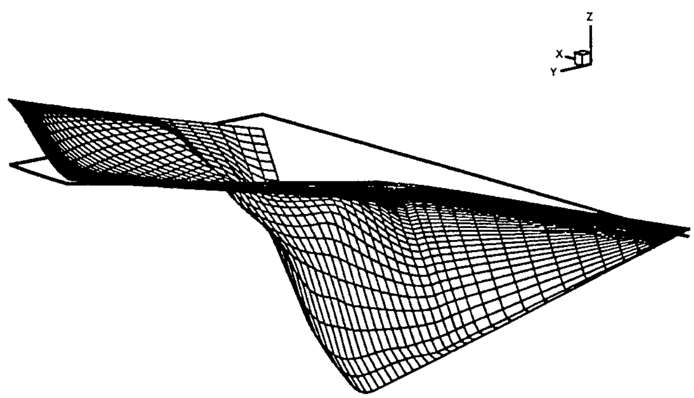


b) Mode Shape Deflections

Figure 13-76. F-16 Wing and Strake Mode Shape 7 Interpolations Using the Multiquadrics Method

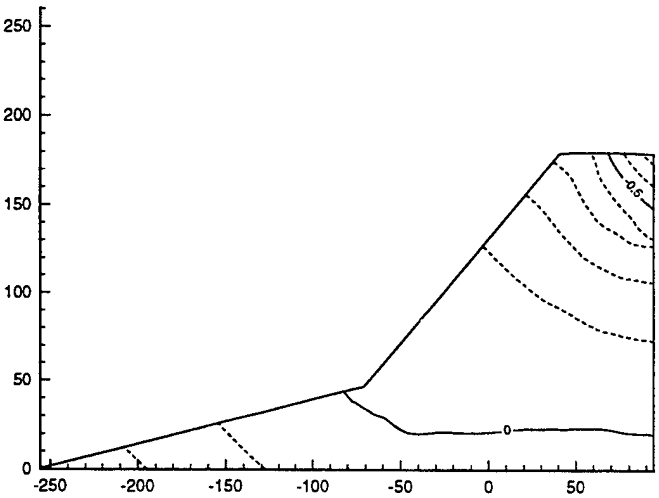


a) Mode Shape Contours

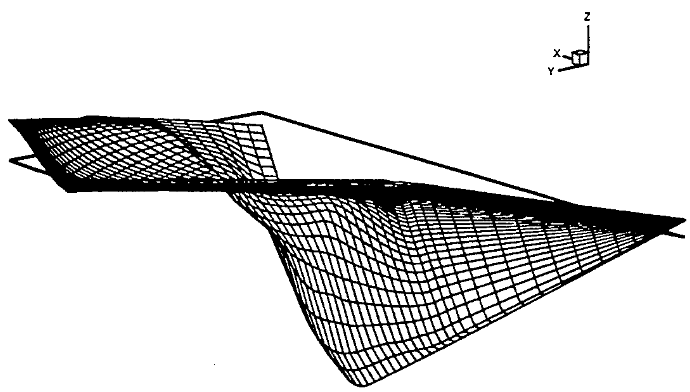


b) Mode Shape Deflections

Figure 13-77. F-16 Wing and Strake Mode Shape 1 Interpolations Using the Thin-Plate Spline Method

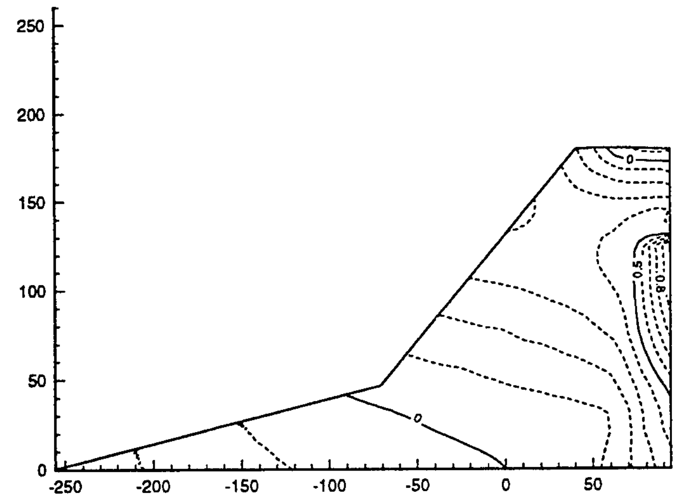


a) Mode Shape Contours

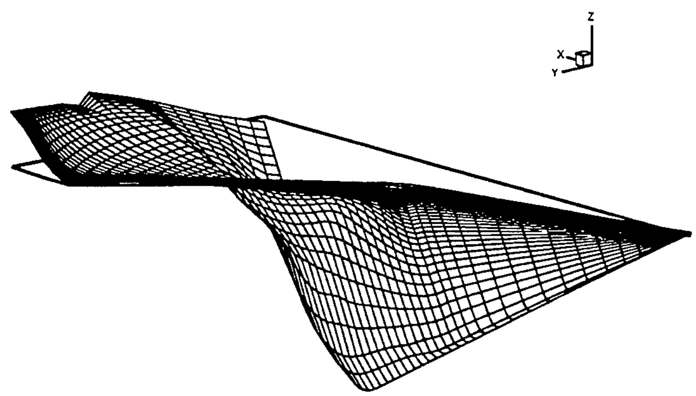


b) Mode Shape Deflections

Figure 13-78. F-16 Wing and Strake Mode Shape 2 Interpolations Using the Thin-Plate Spline Method

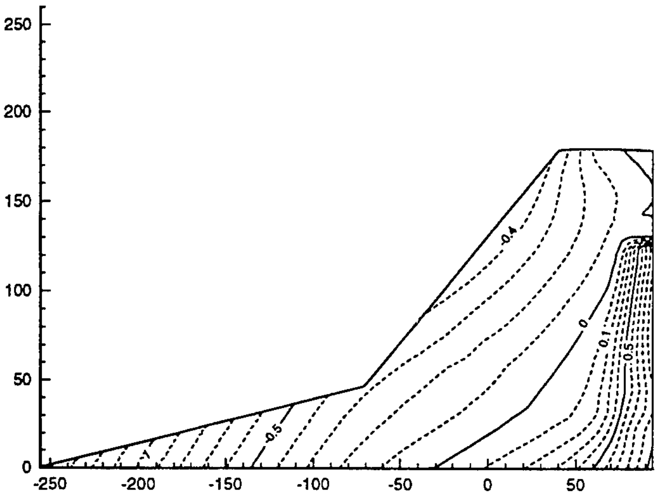


a) Mode Shape Contours

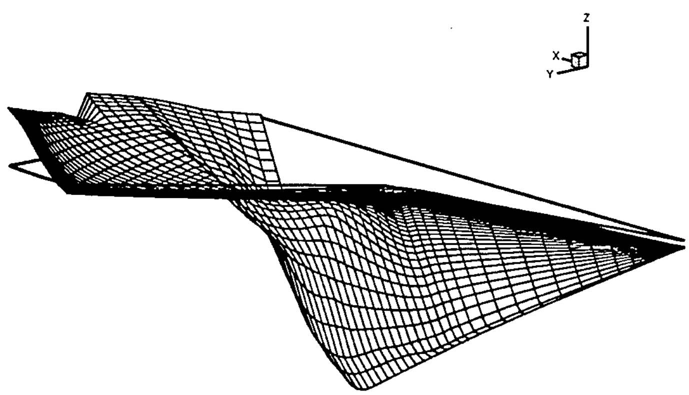


b) Mode Shape Deflections

Figure 13-79. F-16 Wing and Strake Mode Shape 3 Interpolations Using the Thin-Plate Spline Method

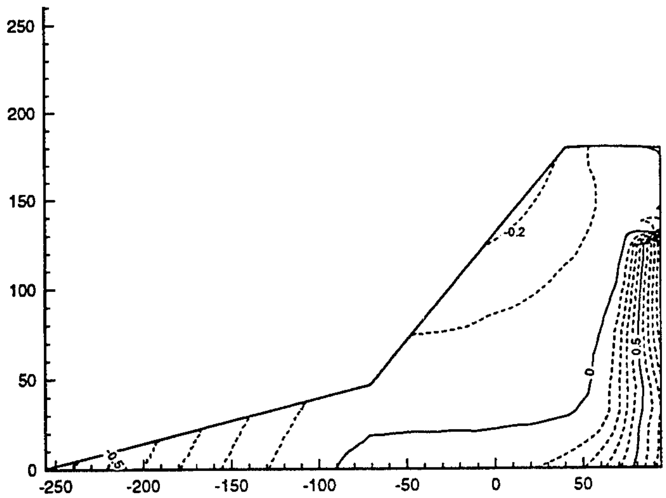


a) Mode Shape Contours

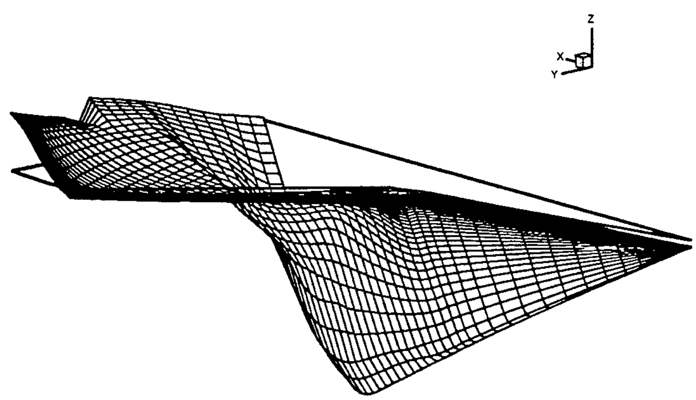


b) Mode Shape Deflections

Figure 13-80. F-16 Wing and Strake Mode Shape 4 Interpolations Using the Thin-Plate Spline Method

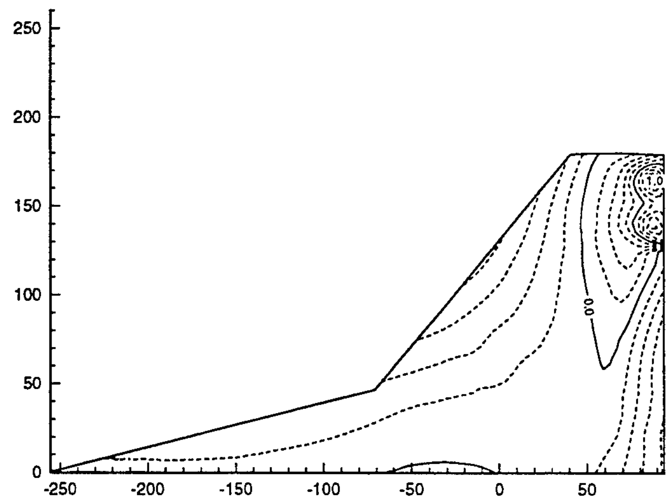


a) Mode Shape Contours

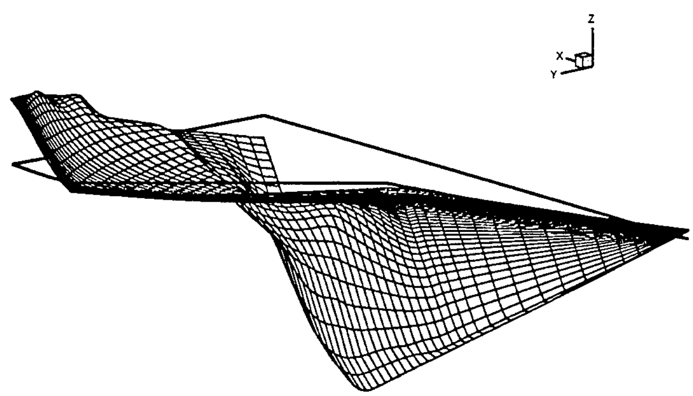


b) Mode Shape Deflections

Figure 13-81. F-16 Wing and Strake Mode Shape 5 Interpolations Using the Thin-Plate Spline Method

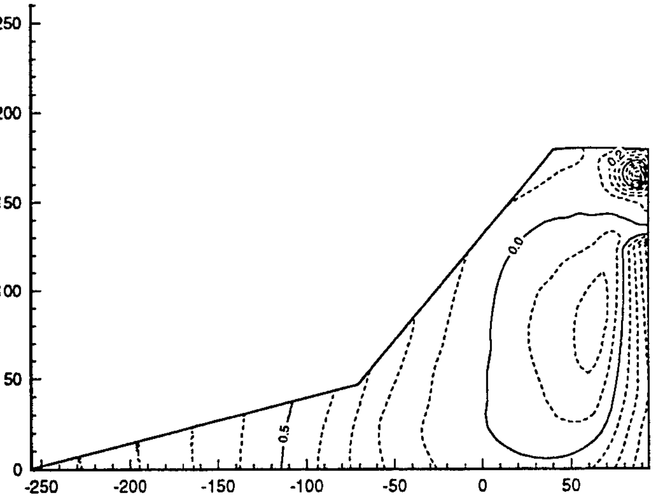


a) Mode Shape Contours

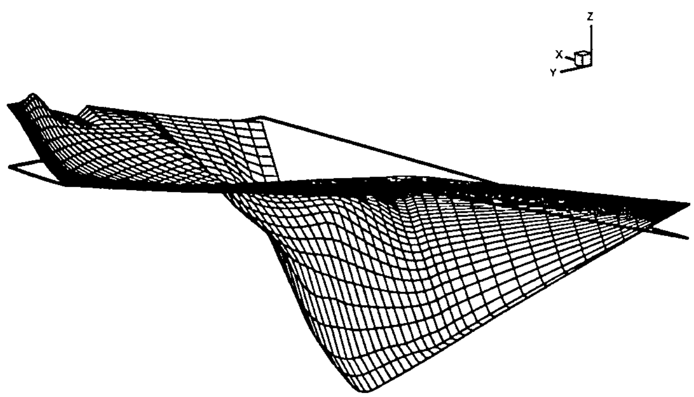


b) Mode Shape Deflections

Figure 13-82. F-16 Wing and Strake Mode Shape 6 Interpolations Using the Thin-Plate Spline Method



a) Mode Shape Contours



b) Mode Shape Deflections

Figure 13-83. F-16 Wing and Strake Mode Shape 7 Interpolations Using the Thin-Plate Spline Method

13.5 F-16 Flexible Wing with Rigid Body

The F-16 flexible wing was examined independently of its rigid body. There was no structural information for the body. The influence coefficient interpolations are shown in Figures 13-84 – 13-87 for the IPS, MQ and TPS methods, respectively.

The F-16 loads computation was perfomed using integration of the pressures from the aerodynamic grid to the structural grid. These load computations are shown in Figures 13-88 and 13-89 for MQ and TPS.

The problem of transforming concentrated loads from one mesh to another can be based on the criteria of conservation of virtual work done by a set of applied loads. If F_a represents a column matrix of given concentrated loads at a set of points P_a, the virtual work done by this set of loads is given by

$$\delta W = F_s^T \delta u_a$$

where δu_a is compatible virtual discretized displacement field corresponding to the set of points P_a.

Consider now a second set of points P_s, not necessarily coincident with the first one. It is possible to define an equivalent set of concentrated loads on this second set such that the virtual work is the same. This condition will guarantee the preservation of total force and moment about any point on the system. If F_s is the set of concentrated forces on the set of points P_s that is equivalent to F_a, then

$$F_s^T \delta u_s = F_a^T \delta u_a$$

where δu_s is a compatible virtual discretized displacement field corresponding to the set of points P_s.

To interpolate the displacement field between the two set of points :

$$u_s = T\, u_a$$

where T is the transformation matrix relating the two displacement sets. Since the virtual displacement fields have to be compatible, they also have to follow the same transformation. Therefore

$$\delta u_s = T\, \delta u_a$$

Using this relation in the expression of the equivalence of virtual work, one gets

$$F_s^T\, T\, \delta u_a = F_a^T\, \delta u_a$$

and

$$F_s^T\, T = F_a^T$$

Finally, the relation between the two sets of forces satisfying conservation of total force and moment is given by

$$F_a = T^T F_s$$

In standard finite element formulation, there is the problem of concentrated loads not applied at nodes. A way to handle the problem is by interpolating those loads to certain nodals points by means of a consistent nodal load transformation. This is basically the method described in the previous section. In order to put that within the finite element context, let us consider the matrix N as being the matrix of shape functions (local interpolation functions for the displacement within an element). Following directly from the general procedure outlined before, if Q is the concentrated force at a point P within an element that is not one of its nodes, then

$$F_n = N^T|_P Q$$

With F_n being the equivalent transformed nodal loads.

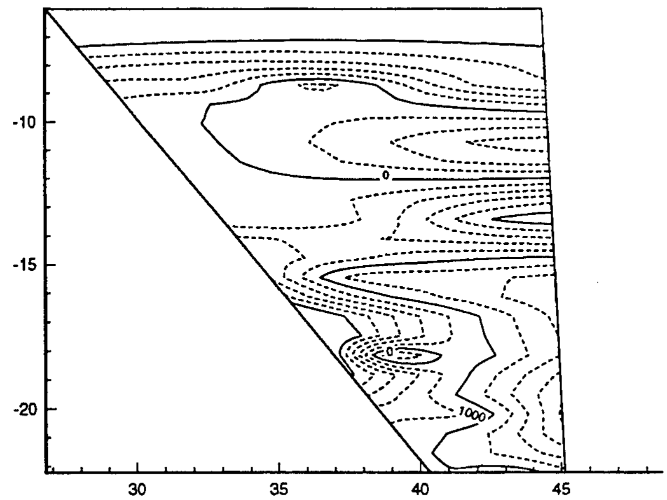


Figure 13-84. F-16 Wing Structural Influence Coefficients Using Infinite-Plate Splines

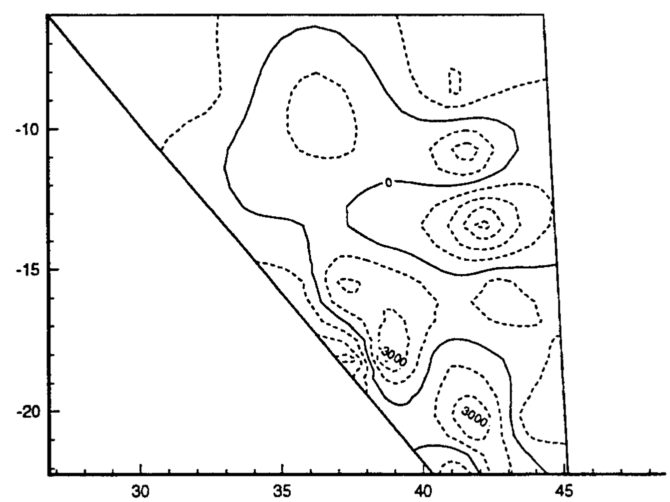


Figure 13-85. F-16 Wing Structural Influence Coefficients Using Multiquadrics

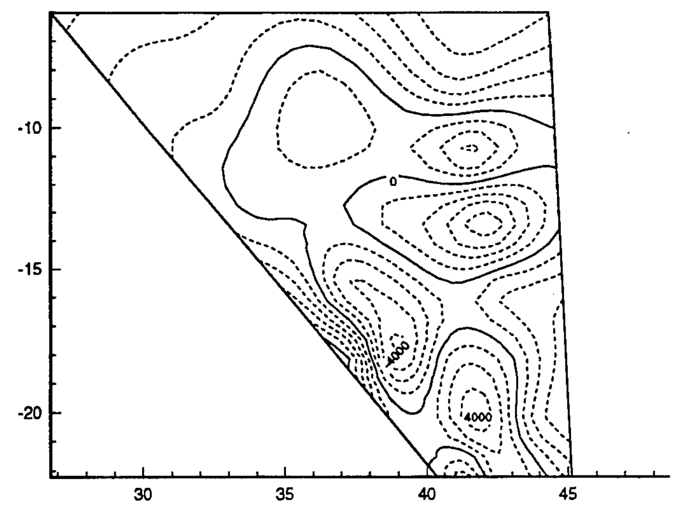


Figure 13-86. F-16 Wing Structural Influence Coefficients Using Thin-Plate Splines

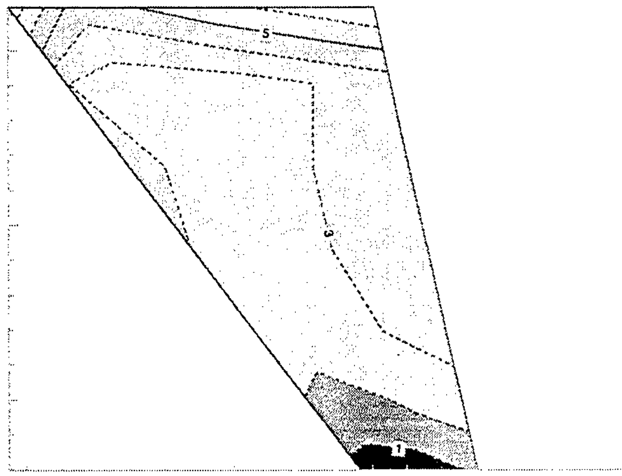


Figure 13-87. F-16 Wing Loads Calculation Using Multiquadrics

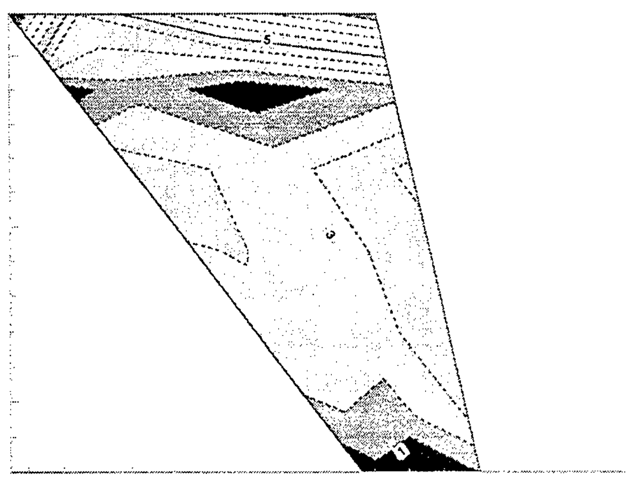


Figure 13-88. F-16 Wing Loads Calculation Using Thin-Plate Splines

14. CONCLUSIONS

From this study, the six methods which were investigated were graded, based upon each characteristic examined. These grades are shown in Table 14.1.

The methods were then applied to realistic applications cases which are current problems being investigated by CFD or aeroelastic methodologies. From these cases, some additional limitations and observed were established, as shown in Table 14.2.

Based on the conclusions drawn from both sets of test cases, the methods can be ranked in the following order:

<u>Three-Dimensional Schemes</u>
Thin-Plate Splines (TPS)
Multiquadrics (MQ)
Infinite-Plate Splines (IPS)
Non-Uniform B-Splines (NUBS)

<u>Two-Dimensional Schemes</u>
Inverse Isoparametric Mapping (IIM)
Finite-Plate Splines (FPS)

For workstation applications, it is recommended that Thin-Plate Splines be used. This method is the most accurate, robust and cost-efficient of all of the methods tested. It is recommended that Finite-Plate Splines only be used on Supercomputers or Massively Parallel Computers because of its large memory requirements to obtain accurate interpolations. Multiquadrics can be applied as long as the grids are single-valued in at least two directions. The Infinite-Plate Spline is recommended only for lifting surface applications, preferrably with the little to no extrapolation required.

Further research is recommended for the Non-Uniform B-Spline and Inverse Isoparametric Mapping Methods. This methods performed very well for limited cases, but need expanding and upgrading to provide robust application to a wide variety of configurations. These recommendations are outlined in the following Chapter.

Application of these methods will vary by configuration. Therefore, it is recommended that several methods be available if many different configurations are to be analyzed within one software package.

Table 14.1 Evaluation of the Algorithm Performance Using the Analytical Test Cases

	IPS	MQ	NUBS	TPS	FPS*	IIM*
Overall Accuracy	B-	A	B	A	A-	A
Robustness	C	A	B	A	A	B
Ease of Use (User Friendliness)	A	B	A	B	A	A
Grid Insensitivity	A-	A	B	A	A	B
Insensitvity to Directional Bias	A	A	A	A	A	A
Magnitude/Amplitude Sensitivity	B	A	A	A	A	A
Sensitivity to Rapid Varying Functions	C	B	B	B	B	A
Extrapolation	C	A	A	A	A	N/A
Diminishing Variation	A	A	A	A	N/A	A
Sensitivity to 3-D Surfaces	B	A	B	A	N/A	N/A
Application to Unstructured or Irregular Grids	B	A	A	A	A	N/A
Computer Memory	C	B	B	B	F	A
Computer Time	C	B	A	B	A	A

* Two-Dimensional Formulation

Table 14.2 Limitations on Methodologies, As Noted During the Applications Test Cases

Method	Limitations
IPS	IPS is extremely slow compared with the other methods. It is also not as accurate as some of the other methods. IPS requires that all three coordinate directions have single-valued grids. This requires the user to partition the configuration accordingly. IPS has some problems with oscillations in the interpolated functions.
MQ	MQ may require scaled or unscaled interpolations. The correlation appears to be that scaled operations are necessary when the grid scales vary widely from the function data scales. In addition, MQ has required for these applications test cases that at least two of the three Cartesian coordinate directions for the grid be single-valued. MQ works very accurately for small extrapolation areas, but distorts the overall function when large extrapolation areas are required.
NUBS	NUBS has some problems with its correlation of the aerodynamic grid points with the structural mesh points. This problem is due to non-DT_NURBS routines, and may be alleviated with the introduction of more accurate algorithms. This inaccuracy results in oscillations in the function contours and failures on some grids.
TPS	TPS may require scaled or unscaled interpolations. The correlation appears to be that scaled operations are necessary when the grid scales vary widely from the function data scales. MQ works very accurately for small extrapolation areas, but distorts the overall function when large extrapolation areas are required.
FPS	The computer memory required to produce an adequate virtual mesh for accurate interpolations exceeded that for a typical computer workstation. This method is recommended only for mainframes or supercomputers with large core memories.
IIM	This method is limited by its two-dimensional formulation, as well as its inability to provide extrapolation. Its search routine has also had some difficulties resolving the location of aerodynamic surface points with respect to their structural counterparts.

15. RECOMMENDATIONS FOR FURTHER STUDY

The Inverse Isoparametric Mapping showed great promise in its two-dimensional applications. This method is the closest simulation of actual finite-element structural analyses. This method will require the following elements to be upgraded if it is to be used in a general applications package:

- extension from two-dimensions to three-dimensions
- elimination of the requirement for structured grids
- accurate interpolation capability
- improved "search" routines to identify grid node linkages

The Non-Uniform B-Splines Method showed excellent promise, but has problems with its search and bivariate interpolation scheme which correlate the known and unknown grid points. It is recommended that these two routines be replaced with higher-order routines to minimize the errors introduced by these two algorithms. NUBS has the added advantage that it can be up-graded to handle IGES and CAD files to more accurately provide surface data to structural and design engineers.

16. ACKNOWLEDGMENTS

This research was funded by the Wright Laboratory Aeromechanics Division (WL/FIM) via contract F33615-94-C-3010. The program technical monitor was Dr. Don Kinsey. The task technical monitor was Capt. Kenneth Moran of the CFD Research Branch of Wright Laboratory.

The authors would like to thank the following people for their contributions to this effort :

- Capt. Kenneth Moran of WL/FIM for his technical support and reviews
- Mr. Larry Huttsell of WL/FIBGE for his help in obtaining the survey results, for permission to use the engine applications test case and for his technical insight into the problems associated with aeroelastic simulations
- Dr. Richard Franke of the Naval Postgraduate School and Dr. Kari Appa of Northrop Corporation for their technical recommendations and help given during code development of the Multiquadrics and Finite Surface Spline methods, respectively.
- Dr. Robert Fithen of the University of New Orleans for his technical explanations and starting code for the Inverse Isoparametric Mapping
- Dr. Robert Ames of the David Taylor Naval Research Center for his help in using the DT_NURBS library, and all other DT_NURBS users who contributed suggestions
- Mr. Ray Kolonay of Wright Laboratory (WL/FIB) for obtaining the structural data for the F-16 and AGARD wing applications test cases.
- Mr. Curt Niebur, Mr. Jeff Hedlund, Dr. Ray Hixon and Mr. Mark Smilor for the assistance in formatting, plotting, coding, and doing "leg-work" for the literature search

17. REFERENCES

1. "Flight Loads Prediction Methods for Aircraft, Volume I: Euler/Navier-Stokes Aeroelastic Method (ENS3DAE) Technical Development Summary", WRDC-TR-3104, 1989.
2. Byun, C. and Guruswamy, G. P., "A Comparative Study of Serial and Parallel Aeroelastic Computations of Wings," NASA Technical Memorandum 108805, January 1994.
3. Robinson, B. A., Batina, J. T., Yang, H. T. Y., "Aeroelastic Analysis of Wings Using the Euler Equations with a Deforming Mesh," *AIAA Journal of Aircraft*, Vol. 28, No. 11, November 1991, pp. 781-788.
4. Guruswamy, G. P., and Byun, C., "Fluid-Structural Interactions Using Navier-Stokes Flow Equations Coupled with Shell Finite Element Structures," AIAA Paper 93-3087, 24th Fluid Dynamics Conference, 6 – 9 July 1993, Orlando, FL.
5. Franke, R., "Scattered Data Interpolation: Tests of Some Methods," *Mathematics of Computations*, Vol. 38, No. 157, Jan. 1982, pp. 181-200.
6. Shan, R., "MPROC3D: User and Program Reference Guide," Draft Version, WL/FIB Contract F33657-90-D-2189, Task 012, October 1993.
7. Walatka, P. P., Clucas, J., McCabe, R. K., Plessel, T., Potter, R., "FAST User Guide", FAST 1.0, Cosmic Program ARC-13316, NASA-Ames, November, 1992.
8. "Tecplot Version 6 User's Manual," Amtec Engineering, Inc., Bellevue, WA, 1994.
9. Turner, E. W., "Interpolation of flexibility Influence Coefficients from One Grid to Another Using Infinite-Plate Splines," Technical Memorandum WRDC-TM-91-000-FIBE, Wright Laboratories, Wright-Patterson AFB, Ohio, July 1991.
10. Harder, R. L., and Demarais, R. N., "Interpolation Using Surface Splines," *AIAA Journal*, vol. 9, no. 2, February 1972, pp. 189 – 191.
11. Franke, R., "Scattered Data Interpolation: Tests of Some Methods," *Mathematics of Computations*, Vol. 38, No. 157, Jan. 1982, pp. 181-200.
12. Kansa, E.J., "Multiquadrics - A Scattered Data Approximation Scheme with Applications to Computational Fluid Dynamics - I: Surface Approximations and Partial Derivative Estimates," *Computers and Mathematics with Applications*, 19(9/9):127-145, 1990.
13. Kansa, E.J., "Multiquadrics - A Scattered Data Approximation Scheme with Applications to Computational Fluid Dynamics - II: Solutions to Parabolic, Hyperbolic and Elliptic Partial Differential Equations," *Computers and Mathematics with Applications*, 19(9/9):147-161, 1990.
14. Appa, K., "Finite-Surface Spline," *Journal of Aircraft*, 26(5):495-496, May 1989.
15. Fithen, R., Private Communication.
16. Yates, E. C., "AGARD Standard Aeroelastic Configurations for Dynamic Response, I. Wing 445.6", AGARD-R-765.
17. Ziada, S., Buhlmann, E. T., Bolleter, U., "Model Tests on Shell Flutter Due to Flow on Both Sides," Journal of Fluids and Structures, Vol. 2:177-196, 1988.
18. Rickets, R. H., Noll, T. E., Whitlow, W., and Huttsell, L. J., "An Overview of Aeroelasticity Studies for the National Aero-Space Plane", AIAA Paper No. 93-1313, April, 1993.

Appendix A: Report Review List

Bibliography DataBase File: "Interface"
May 2, 1996

References

[Ames1995] R. Ames. Fitsurf 6.0. Private correspondence, 1995.

[Anon1988] Anon. *Connectivity between Aerodynamic and Structural Models / Section 8.3 ASTROS Theoretical Manual.* Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio, 1988.

[Anon1995] Anon. *DT_NURBS Spline Geometry Subprogram Library User's Manual.* Boeing Computer Services, version 2.3 edition, February 1995. pp. 3-1 – 4-5.

[AnthonyCox1987] G. T. Anthony and M. G. Cox. The fitting of extremely large data sets by bivariate splines. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 5 – 20. Clarendon Press, Oxford, 1987.

[Appa1989] Kari Appa. Finite-surface spline. *Journal of Aircraft*, 26(5):495 – 496, May 1989.

[Appa1991] Kari Appa. Recent advances in maneuver loads analysis. *Computer Methods in Applied Mechanics and Engineering*, pages 693 – 717, September 1991.

[AppaYankulichCowan1985] Kari Appa, Micheal Yankulich, and David L. Cowan. The determination of load and slope transformation matrices for aeroelastic analyses. *Journal of Aircraft*, 22(8):734 – 736, August 1985.

[Argyris1966] J. H. Argyris. Matrix displacement analysis of plates and shells – prolegomena to a general theory, part I. *Ingenieur-Archiv*, 35(2):102 – 142, 1966.

[Atteia1970] M. Atteia. Fonctions 'spline' et noyaux reproduisants d'Aronszajn-Bergman. *Revue Française d'Informatique et de Recherche Opérationelle*, R-3:31 – 43, 1970.

[BarnhillPiperRescorla1987] R. E. Barnhill, B. R. Piper, and K. L. Rescorla. Interpolation to arbitrary data on a surface. In Gerald E. Farina, editor, *Geometric Modeling: Algorithms and New Trends*, pages 281 – 289. Society for Industrial and Applied Mathematics, 1987.

[Bathe1982] K. J. Bathe. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, Inc., Englewood Cliff, New Jersey, 1982.

[BeatsonZiegler1985] R. K. Beatson and Z. Ziegler. Monotonicity preserving surface interpolation. *SIAM Journal of Numerical Analysis*, 22(2):401 – 411, April 1985.

[Bezier1993] Pierre E. Bezier. The first years of CAD/CAM and the UNISURF CAD system. In Les Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 13 – 26. Academic Press Limited, San Diego, California, 1993.

[BinevJetter1992] P. Binev and K. Jetter. Estimating the condition number for multivariate interpolation problems. In Dietrich Braess and Larry L. Schumaker, editors, *Numerical Methods of Approximation Theory, vol. 9*, pages 41 – 52. Birkhaeuser Verlag, Basel, 1992.

[BirkhoffGarabedian1960] Garrett Birkhoff and Henry L. Garabedian. Smooth surface interpolation. *Journal of Mathematics and Physics*, 39:258 – 268, 1960.

[Blair1994] Max Blair. A unified aeroelastic surface formulation. In *Proceedings of the 35th Structures, Structural Dynamics and Materials Conference, Hilton Head, South Carolina*, pages 1276 – 1284, April 18 – 20 1994. AIAA Paper No. 94-1471.

[BlessMoerder1995] R. R. Bless and D. D. Moerder. A computationally efficient method for multidimensional data interpolation. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Baltimore, Maryland*, August 7-10 1995.

[Borland1986] C. J. Borland. *XTRAN3S – Technical Development Summary.* Boeing Military Airplane Company, Seattle, Washington, January 1986.

[Bosworth1987] K. W. Bosworth. Shape constraint curve and surface fitting. In Gerald E. Farina, editor, *Geometric Modeling: Algorithms and New Trends*, pages 247 – 263. Society for Industrial and Applied Mathematics, 1987.

[ByunGuruswamy1994] Chansup Byun and Guru P. Guruswamy. Wing-body aeroelasticity using finite-difference fluid/finite-element structural equations on parallel computers. In *Proceedings of the 35th Structures, Structural Dynamics, and Materials Conference, Hilton Head, South Carolina*, pages 1356 – 1365. AIAA, April 18 – 20 1994. AIAA Paper No. 94-1487.

[ChuiLai1987] Charles K. Chui and M. Lai. On multivariate vertex splines and applications. In C.K. Chui, L.L. Schumaker, and F.I. Utreras, editors, *Topics in Multivariate Approximation*, pages 19 – 36. Academic Press, Inc., 1987.

[CookMalkusPlesha1989] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis*. John Wiley and Sons, New York, 3rd edition, 1989.

[Coull1965] A. Coull. A direct-stress analysis of orthotropic cantilever plates. *Journal of Applied Mechanics*, 32:67 – 70, March 1965.

[Cox1993] Maurice G. Cox. Algorithms for spline curves and surfaces. In Les Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, chapter 4, pages 51 – 76. Academic Press Limited, San Diego, California, 1993.

[Dahmen1987] W. Dahmen. Subdivision algorithms – recent results, some extensions and further developments. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 21 – 49. Clarendon Press, Oxford, 1987.

[Done1965] G. T. S. Done. Interpolation of mode shapes: A matrix scheme using two-way spline curves. *The Aeronautical Quarterly*, pages 333 – 349, November 1965.

[Duchon1976] Jean Duchon. Fonctions-spline à énergie invariante par rotation. Technical Report R. R. No. 27, Université de Grenoble, January 1976.

[Duchon1977] Jean Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions of Several Variables, Oberwolfach 1976*, pages 85 – 100. Springer-Verlag, Berlin, 1977.

[Farin1988] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, Inc., Boston, 1988.

[FergusonMastroBlakely1989] D. Ferguson, R. Mastro, and R. Blakely. Modeling and analysis of aerodynamic data. Technical Report AIAA Paper No. 89-0476, AIAA, 370 L'Enfant Promenade, S.W., Washington D.C. 20024, January 1989.

[Fithen1995] R. Fithen. Interp. Private correspondence, 1995.

[Foley1987] T. A. Foley. Weighted bicubic spline interpolation to rapidly varying data. *ACM Transactions on Graphics*, 6(1):1 – 18, January 1987.

[Fontanella1987] Ferruccio Fontanella. Shape preserving surface interpolation. In C.K. Chui, L.L. Schumaker, and F.I. Utreras, editors, *Topics in Multivariate Approximation*, pages 63 – 78. Academic Press, Inc., 1987.

[Franke1982] Richard Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38(157):181 – 200, January 1982.

[Franke1987] Richard Franke. Recent advances in the approximation of surfaces from scattered data. In C.K. Chui, L.L. Schumaker, and F.I. Utreras, editors, *Topics in Multivariate Approximation*, pages 79 – 98. Academic Press, Inc., 1987.

[FrankeSchumaker1987] R. Franke and L. L. Schumaker. A bibliography of multivariate approximation. In C.K. Chui, L.L. Schumaker, and F.I. Utreras, editors, *Topics in Multivariate Approximation*, pages 275 – 335. Academic Press, Inc., 1987.

[GeaChowChang1992] Lie-Mine Gea, Chuen-Yen Chow, and I-Chung Chang. Transonic aeroelastic analysis for rotor blades. *Journal of Aircraft*, 29(3):477 – 484, May-June 1992.

[Giles1986] Gary Giles. Equivalent plate analysis of aircraft wing box structures with general planform geometry. *Journal of Aircraft*, 23(11):859 – 863, November 1986.

[Giles1989] Gary L. Giles. Further generalization of an equivalent plate representation for aircraft structural analysis. *Journal of Aircraft*, 26(1):67 – 74, January 1989.

[Greville1969] T. N. E. Greville. Introduction to spline functions. In T. N. E. Greville, editor, *Theory and Applications of Spline Function*, pages 1 – 36, New York, 1969. Mathematics Research Center, Academic Press.

[Guruswamy1994] G. P. Guruswamy. A comparative study of serial and parallel aeroelastic computations of wings. Technical Memorandum 108805, NASA Ames Research Center, Moffett Field, California, January 1994.

[GuruswamyByun1993] G. P. Guruswamy and C. Byun. Fluid-structural interactions using navier-stokes flow equations coupled with shell finite element structures. In *Proceedings of the 24th Fluid Dynamics Conference*. AIAA, July 6 – 9 1993. AIAA Paper 93-3087.

[GuruswamyByun1995] G. P. Guruswamy and C. Byun. Direct coupling of euler flow equations with plate finite element structures. *AIAA Journal*, 33(2):375 – 377, February 1995.

[HarderDesmarais1972] R. L. Harder and R. N. Desmarais. Interpolation using surface splines. *Journal of Aircraft*, 9(2):189 – 191, 1972.

[Hardy1971] Rolland L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research*, 76(8):1905 – 1915, March 1971.

[Hardy1990] Rolland L. Hardy. Theory and applications of the multiquadric biharmonic-method. *Computers and Mathematics with Applications*, 19(8/9):163 – 208, 1990.

[HardyNelson1986] Rolland L. Hardy and Stuart A. Nelson. A multiquadric-biharmonic representation and approximation of disturbing potential. *Geophysical Research Letters*, 13(1):18 – 21, January 1986.

[Jones1987] A. K. Jones. Shape control of curves and surfaces through constrained optimization. In Gerald E. Farina, editor, *Geometric Modeling: Algorithms and New Trends*, pages 265 – 279. Society for Industrial and Applied Mathematics, 1987.

[Kansa1990a] E. J. Kansa. Multiquadrics – a scattered data approximation scheme with applications to computational fluid-dynamics – I: surface approximations and partial derivative estimates. *Computers and Mathematics with Applications*, 19(8/9):127 – 145, 1990.

[Kansa1990b] E. J. Kansa. Multiquadrics – a scattered data approximation scheme with application to computational fluid-dynamics – II: solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers and Mathematics with Applications*, 19(8/9):147 – 161, 1990.

[Kreyszig1988] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, New York, 1988.

[LancasterSalkauskas1986] Peter Lancaster and Kestutis Salkauskas. *Curve and Surface Fitting*, chapter 4, pages 97 – 111. Academic Press, London, 1986.

[LeMehaute1987] Alain J. Y. Le Mehaute. Unisolvent interpolation in IR^n and the simplicial polynomial finite element method. In C.K. Chui, L.L. Schumaker, and F.I. Utreras, editors, *Topics in Multivariate Approximation*, pages 19 – 36. Academic Press, Inc., 1987.

[LeeLoellbach1988] K.D. Lee and J.M. Loellbach. Geometry-adaptive surface grid generation using parametric projection. *Journal of Aircraft*, 26(2):162 – 166, February 1988.

[Loscalzo1969] Rank R. Loscalzo. An introduction to the application of spline functions to initial value problems. In T. N. E. Greville, editor, *Theory and Applications of Spline Function*, pages 37 – 64, New York, 1969. Mathematics Research Center, Academic Press.

[MSC/NASTRAN1987] Anon./NASTRAN. *Interconnection of Structure with Aerodynamics / Section 2.4 of MCS/NASTRAN Handbook for Aeroelastic Analysis Vol. 1*. MacNeal-Schwendler Corporation., 815 Colorado Blvd., Los Angeles, California, September 1987.

[Meirovitch1967] L. Meirovitch. *Analytical Methods in Vibrations*. MacMillian Publishing Co., New York, 1967.

[MontefuscoCasciola1989] Laura Montefusco and Giulio Casciola. Algorithm 677 C^1 surface interpolation. *ACM Transactions on Mathematical Software*, 15(4):365 – 374, December 1989.

[MurtiValliappan1986] V. Murti and S. Valliappan. Numerical inverse isoparametric mapping in remeshing and nodal quantity contouring. *Computers and Structures*, 22(6):1011 – 1021, 1986.

[Nash1952] W.A. Nash. Several approximate analyses of the bending of a rectangular cantilever plate by uniform normal pressure. *Journal of Applied Mechanics*, Unknown(Unknown):33 – 36, March 1952.

[ObayashiGuruswamy1994] S. Obayashi and G.P. Guruswamy. Convergence acceleration of an aeroelastic navier-stokes solver. In *25th AIAA Fluid Dynamics Conference*. AIAA, June 20 – 23 1994. AIAA Paper No. 94-2268.

[Pidaparti1992] R. M. V. Pidaparti. Structural and aerodynamic data transformation using inverse isoparametric mapping. *Journal of Aircraft*, 29(3):507 – 509, 1992.

[PieglTiller1987] L. Piegl and W. Tiller. Curve and surface constructions using rational b-splines. *Computer Aided Design*, 19(9):485 – 495, November 1987.

[PittmanGiles1988] James L. Pittman and Gary L. Giles. Combined, nonlinear aerodynamic and structural method for the aeroelastic design of a three-dimensional wing in supersonic flow. Technical Report AIAA Paper No. 88-1769, AIAA, 370 L'Enfant Promenade, S.W., Washington D.C. 20024, 1988.

[Powell1987] M. J. D. Powell. Radial basis functions for multivariate interpolation: a review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143 – 167. Clarendon Press, Oxford, 1987.

[Powell1992] M. J. D. Powell. Tabulation of thin plate splines on a very fine two-dimensional grid. In Dietrich Braess and Larry L. Schumaker, editors, *Numerical Methods of Approximation Theory, vol. 9*, pages 221 – 244. Birkhaeuser Verlag, Basel, 1992.

[Riesenfeld1993] R. Riesenfeld. Modeling with nurbs curves and surfaces. In *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 77 – 97. Academic Press Limited, San Diego, California, 1993.

[Risler1992] J.-J. Risler. *Mathematical Methods for CAD.* Cambridge University Press, 1992.

[RobinsonBatinaYang1991] B. Robinson, J. Batina, and H. Yang. Aeroelastic analysis of wings using the Euler equations for a deforming mesh. *Journal of Aircraft*, 28(11):781 – 788, November 1991.

[Rodden1959] William P. Rodden. Further remarks on matrix interpolation of flexibility influence coefficients. *Journal of Aerospace Sciences*, 26:760 – 761, November 1959.

[RoddenMcGrewKalman1972] William Rodden, Jean A. McGrew, and Terez P. Kalman. Comment on "Interpolation using surface splines". *Journal of Aircraft*, 9(12):869 – 871, December 1972.

[Schmitt1956] Alfred F. Schmitt. A least squares matrix interpolation of flexibility influence coefficients. *Journal of Aeronautical Sciences*, page 980, October 1956.

[Shan1993] R. Shan. MPROC3D: User's and program reference guide. Unpublished report, 1993.

[SmithCampell1990] Leigh Smith and Richard Campell. Method for designing of flexible transonic wings. Technical Paper 2045, NASA Langley Research Center, Hampton,Virginia, November 1990.

[TatumGiles1987] K.E. Tatum and G.L. Giles. Integrating nonlinear aerodynamic and structural analysis for a complete fighter configuration. Technical Report AIAA Paper No. 87-2863, AIAA, 370 L'Enfant Promenade, S.W., Washington D.C. 20024, September 1987.

[TatumGiles1989] Kenneth Tatum and Gary L. Giles. Integrating nonlinear aerodynamic and structural analysis for a complete fighter configuration. *Journal of Aircraft*, 25(12):1150 – 1154, December 1989.

[Tiller1983] W. Tiller. Rational b-splines for curve and surface representation. *IEEE Computer Graphics and Applications*, pages 61 – 69, September 1983.

[Turner1991] E. W. Turner. Interpolation of flexibility influence coefficients from one grid to another using infinite surface splines. Technical Memorandum WRDC-TM-91-000-FIBE, Wright Laboratories, Wright-Patterson Air Force Base, Ohio, July 1991.

[Turner1991u] E. W. Turner. Flexibility influence coefficients matrix interpolation from one grid to another. Unpublished notes, 1991.

[Utreras1987] Florencio Utreras. Constrained surface construction. In C. K. Chui, L. L. Schumaker, and F. I. Utreras, editors, *Topics in Multivariate Approximation*, pages 223 – 254. Academic Press, Inc., 1987.

[WalshAhlbergNilson1962] J.L. Walsh, J.H. Ahlberg, and E.N. Nilson. Best approximation of the spline fit. *Journal of Mathematics and Mechanics*, 11(2):225 – 233, 1962.

[WhitlowBennett1982] Woodrow Whitlow and Robert M. Bennett. Application of a transonic potential flow code to the static aerodynamic analysis of three-dimensional wings. Technical Report AIAA Paper No. 82-0689, AIAA, 370 L'Enfant Promenade, S.W., Washington D.C. 20024, 1982.

[deBoor1962] Carl de Boor. Bicubic spline interpolation. *Journal of Mathematics and Physics*, 41:213 – 218, February 1962.

[deBoor1993] Carl de Boor. B(asic)-spline basics. In Les Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, chapter 3, pages 27 – 49. Academic Press Limited, San Diego, California, 1993.

[deCasteljau1993] Paul de Casteljau. Polar forms for curve and surface modeling as using at Citroen. In Les Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, chapter 1, pages 1 – 12. Academic Press Limited, San Diego, California, 1993.

[unknown1] unknown. Criteria for curve and surface fitting. This was on pages 127-131 of some unknown source!

[unknown2] unknown. Surface interpolation by tensor product and blending methods. This was on pages 153-178 of some unknown source!

[unknown3] unknown. Surface splines. This was on pages 245-271 of some unknown source!

APPENDIX B - SUMMARY OF THE ANALYTICAL TEST CASES

Analytical Test Category I - Regular Interpolation

Number of Test Cases : 138

Test Category Description : A rectangular, Cartesian grid is used for both the structural and aerodynamic grids. Interpolation of various functions are examined: zero-gradient functions (constants), linearly varying functions, sinusoidally varying functions. The last set of functions includes superposition of two frequencies to create an irregular pattern. This test will provide the basic characteristics of the interpolation scheme and will be used for general debugging.

Test Case Matrix

Grids are defined by streamwise $\times$ spanwise elements. A is the amplitude or function value.

The first set of test cases checks the accuracy of the algorithms in the streamwise direction.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
1a	10×10 Constant Spacing	10×10 Constant Spacing	Constant A=5.0	Constant A=0.0
1b			Linear A=5.0-0.0	
1c			Sinusoidal, 1 Cycle, A=2.0	
1d		50×20 Clustered at Root and Leading Edge	Constant A=5.0	
1e			Linear A=5.0-0.0	
1f			Sinusoidal, 1 Cycle, A=2.0	
1g		50×20 Clustered at Tip and Trailing Edge	Constant A=5.0	
1h			Linear A=5.0-0.0	
1i			Sinusoidal, 1 Cycle, A=2.0	
1j		50×20 Clustered at all Edges	Constant A=5.0	
1k			Linear A=5.0-0.0	
1l			Sinusoidal, 1 Cycle, A=2.0	
1m		10×10 Constant Spacing	Constant A=50.0	

1n			Linear A=1.01-1.02	
1o			Linear A=0.01 - 0.02	
1p	50×10 Constant Spacing	70×20 Constant Spacing	Sinusoidal, 3 Cycles, A=2.0	
1q	50×10 Constant Spacing	100×20 Constant Spacing	Sinusoidal, 5 Cycles, A=2.0	
1r	150×10 Constant Spacing	200×20 Constant Spacing	Sinusoidal, 7 Cycles, A=2.0	
1s	50×10 Constant Spacing	70×20 Constant Spacing	Sinusoidal, 3 Cycles, A=.02	
1t	50×10 Constant Spacing	70×20 Constant Spacing	Sinusoidal, 5 Cycles, A=.02	
1u	150×10 Constant Spacing	200×20 Constant Spacing	Sinusoidal, 7 Cycles, A=.02	
1a1	10×10 Constant Spacing	10×10 Constant Spacing	Constant A=.05	
1b1			Linear A=.05-0.0	
1c1			Sinusoidal, 1 Cycle, A=.02	
1d1		50×20 Clustered at Root and Leading Edge	Constant A=.05	
1e1			Linear A=.05-0.0	
1f1			Sinusoidal, 1 Cycle, A=.02	
1g1		50×20 Clustered at Tip and Trailing Edge	Constant A=.05	
1h1			Linear A=.05-0.0	
1i1			Sinusoidal, 1 Cycle, A=.02	
1j1		50×20 Clustered at all Edges	Constant A=.05	
1k1			Linear A=.05-0.0	
1l1			Sinusoidal, 1 Cycle, A=.02	

The second set of test cases checks the accuracy of the algorithms in the spanwise direction. This examines the algorithms for the possibility of bias.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
1a2	10×10 Constant Spacing	10×10 Constant Spacing	Constant A=0.0	Constant A=5.0
1b2				Linear A=5.0-0.0
1c2				Sinusoidal, 1 Cycle, A=2.0
1d2		20×50 Clustered at Root and Leading Edge		Constant A=5.0
1e2				Linear A=5.0-0.0
1f2				Sinusoidal, 1 Cycle, A=2.0
1g2		20×50 Clustered at Tip and Trailing Edge		Constant A=5.0
1h2				Linear A=5.0-0.0
1i2				Sinusoidal, 1 Cycle, A=2.0
1j2		20×50 Clustered at all Edges		Constant A=5.0
1k2				Linear A=5.0-0.0
1l2				Sinusoidal, 1 Cycle, A=2.0
1m2		10×10 Constant Spacing		Constant A=50.0
1n2				Linear A=1.01-1.02
1o2				Linear A=0.01 - 0.02
1p2	10×50 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 3 Cycles, A=2.0
1q2	10×50 Constant Spacing	20×100 Constant Spacing		Sinusoidal, 5 Cycles, A=2.0
1r2	10×150 Constant Spacing	20×200 Constant Spacing		Sinusoidal, 7 Cycles, A=2.0

1s2	10×150 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 3 Cycles, A=.02
1t2	10×50 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 5 Cycles, A=.02
1u2	10×150 Constant Spacing	20×200 Constant Spacing		Sinusoidal, 7 Cycles, A=.02
1a12	10×10 Constant Spacing	10×10 Constant Spacing		Constant A=.05
1b12				Linear A=.05-0.0
1c12				Sinusoidal, 1 Cycle, A=.02
1d12		20×50 Clustered at Root and Leading Edge		Constant A=.05
1e12				Linear A=.05-0.0
1f12				Sinusoidal, 1 Cycle, A=.02
1g12		20×50 Clustered at Tip and Trailing Edge		Constant A=.05
1h12				Linear A=.05-0.0
1i12				Sinusoidal, 1 Cycle, A=.02
1j12		20×50 Clustered at all Edges		Constant A=.05
1k12				Linear A=.05-0.0
1l12				Sinusoidal, 1 Cycle, A=.02

The next series of test cases examines the capability of the algorithms in interpolating shell structures. The leading and trailing edges of the grids are level at 0.0, while the thickness (z-parameter) varies from 0.0 to 0.2.

Case Number	**Original Grid**	**Interpolation Grid**	**Streamwise Function Description**	**Spanwise Function Description**
1aa	10×10 Constant Spacing	10×10 Constant Spacing	Constant A=5.0	Constant A=0.0
1bb			Linear A=5.0-0.0	

1cc			Sinusoidal, 1 Cycle, A=2.0	
1dd		20×50 Clustered at Root and Leading Edge	Constant A=5.0	
1ee			Linear A=5.0-0.0	
1ff			Sinusoidal, 1 Cycle, A=2.0	
1gg		20×50 Clustered at Tip and Trailing Edge	Constant A=5.0	
1hh			Linear A=5.0-0.0	
1ii			Sinusoidal, 1 Cycle, A=2.0	
1jj		20×50 Clustered at all Edges	Constant A=5.0	
1kk			Linear A=5.0-0.0	
1ll			Sinusoidal, 1 Cycle, A=2.0	
1mm		10×10 Constant Spacing	Constant A=50.0	
1nn			Linear A=1.01-1.02	
1oo			Linear A=0.01 - 0.02	
1pp	10×50 Constant Spacing	20×70 Constant Spacing	Sinusoidal, 3 Cycles, A=2.0	
1qq	10×50 Constant Spacing	20×100 Constant Spacing	Sinusoidal, 5 Cycles, A=2.0	
1rr	10×150 Constant Spacing	20×200 Constant Spacing	Sinusoidal, 7 Cycles, A=2.0	
1ss	10×150 Constant Spacing	20×70 Constant Spacing	Sinusoidal, 3 Cycles, A=.02	
1ttt	10×50 Constant Spacing	20×70 Constant Spacing	Sinusoidal, 5 Cycles, A=.02	
1uu	10×150 Constant Spacing	20×200 Constant Spacing	Sinusoidal, 7 Cycles, A=.02	

1bb2	10×10 Constant Spacing	10×10 Constant Spacing	Constant A=0.0	Linear A=5.0-0.0
1cc2				Sinusoidal, 1 Cycle, A=2.0
1dd2		20×50 Clustered at Root and Leading Edge		Constant A=5.0
1ee2				Linear A=5.0-0.0
1ff2				Sinusoidal, 1 Cycle, A=2.0
1gg2		20×50 Clustered at Tip and Trailing Edge		Constant A=5.0
1hh2				Linear A=5.0-0.0
1ii2				Sinusoidal, 1 Cycle, A=2.0
1jj2		50×20 Clustered at all Edges		Constant A=5.0
1kk2	.			Linear A=5.0-0.0
1ll2				Sinusoidal, 1 Cycle, A=2.0
1nn2		10×10 Constant Spacing		Linear A=1.01-1.02
1oo2				Linear A=0.01 - 0.02
1pp2	10×50 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 3 Cycles, A=2.0
1qq2	10×50 Constant Spacing	20×100 Constant Spacing		Sinusoidal, 5 Cycles, A=2.0
1rr2	10×150 Constant Spacing	20×200 Constant Spacing		Sinusoidal, 7 Cycles, A=2.0
1ss2	10×150 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 3 Cycles, A=.02
1tt2	10×50 Constant Spacing	20×70 Constant Spacing		Sinusoidal, 5 Cycles, A=.02
1uu2	10×150 Constant Spacing	20×200 Constant Spacing		Sinusoidal, 7 Cycles, A=.02

This final set of data combines streamwise and spanwise functions for both plates and shells.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
1A	10×10 Constant Spac. Plate	10×10 Constant Spac. Plate	Linear A=5.0-0.0	Linear A=5.0-0.0
1B			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1C		20×50 Clustered at all Edges Plate	Linear A=5.0-0.0	Constant A=5.0
1D			Linear A=5.0-0.0	Linear A=5.0-0.0
1E			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
1F			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1G			Linear A=1.01-1.02	Linear A=1.01-1.02
1H			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0
1I	10×50 Constant Space Plate	20×70 Constant Space Plate	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
1J	50×50 Constant Space Plate	70×100 Constant Space Plate	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
1K	150×150 Constant Space Plate	200×200 Constant Space Plate	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02
1A1	10×10 Constant Space Shell	10×10 Constant Space Shell	Linear A=5.0-0.0	Linear A=5.0-0.0
1B1			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1C1		20×50 Clustered at all Edges Shell	Linear A=5.0-0.0	Constant A=5.0
1D1			Linear A=5.0-0.0	Linear A=5.0-0.0
1E1			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
1F1			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1G1			Linear A=1.01-1.02	Linear A=1.01-1.02
1H1			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0

1I1	10×50 Constant Space Shell	20×70 Constant Space Shell	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
1J1	50×50 Constant Space Shell	70×100 Constant Space Shell	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
1K1	150×150 Constant Space Shell	200×200 Constant Space Shell	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02
1A2	10×10 Constant Space Plate	10×10 Constant Space Shell	Linear A=5.0-0.0	Linear A=5.0-0.0
1B2			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1C2		20×50 Clustered at all Edges Shell	Linear A=5.0-0.0	Constant A=5.0
1D2			Linear A=5.0-0.0	Linear A=5.0-0.0
1E2			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
1F2			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
1G2			Linear A=1.01-1.02	Linear A=1.01-1.02
1H2			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0
1I2	10×50 Constant Space Plate	20×70 Constant Space Shell	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
1J2	50×50 Constant Space Plate	70×100 Constant Space Shell	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
1K2	150×150 Constant Space Plate	200×200 Constant Space Shell	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02

Analytical Test Category II - Irregular Structural Grids

Number of Test Cases : 28

Test Category Description : This category confirms that no irregularities are introduced to the interpolations when an irregular structural grid is used. The structural grid varies in both spanwise and streamwise functions along a given "cut" (see figure below). A rectangular, Cartesian grid is used for the aerodynamic grids. Interpolation of various functions are examined : zero-gradient functions (constants), linearly varying functions, sinusoidally varying functions. This set of functions includes superposition of two frequencies to create an irregular pattern.

Test Case Matrix

Grids are defined by streamwise × spanwise elements. A is the amplitude or function value.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
2A	Grid 1	10 × 10 Constant Spac. Plate	Constant A=1.0	Constant A=1.0
2B			Constant A=1.0	Linear A=1.0-0.0
2C			Linear A=1.0-0.0	Constant A=1.0
2D			Linear A=1.0-0.0	Linear A=5.0-0.0
2E			Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 1 Cycle, A=1.0
2F		50 × 50 Clustered Plate	Constant A=1.0	Constant A=1.0
2G			Constant A=1.0	Linear A=1.0-0.0
2H			Linear A=1.0-0.0	Constant A=1.0
2I			Linear A=1.0-0.0	Linear A=5.0-0.0
2J			Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 1 Cycle, A=1.0
2K			Constant A=0.005	Constant A=0.005
2L			Linear A=0.005-0.	Linear A=-0.005-0
2M			Linear A=0.005-0	Linear A=0.-0.005
2N			Sinusoidal, 1 Cycle, A=0.001	Sinusoidal, 1 Cycle, A=0.001
2A1	Grid 1	10 × 10 Constant Spac. Shell	Constant A=1.0	Constant A=1.0
2B1			Constant A=1.0	Linear A=1.0-0.0
2C1			Linear A=1.0-0.0	Constant A=1.0
2D1			Linear A=1.0-0.0	Linear A=5.0-0.0
2E1			Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 1 Cycle, A=1.0
2F1		50 × 50 Clustered Shell	Constant A=1.0	Constant A=1.0

2G1			Constant A=1.0	Linear A=1.0-0.0
2H1			Linear A=1.0-0.0	Constant A=1.0
2I1			Linear A=1.0-0.0	Linear A=5.0-0.0
2J1			Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 1 Cycle, A=1.0
2K1			Constant A=0.005	Constant A=0.005
2L1			Linear A=0.005-0.	Linear A=-0.005-0
2M1			Linear A=0.005-0	Linear A=0.-0.005
2N1			Sinusoidal, 1 Cycle, A=0.001	Sinusoidal, 1 Cycle, A=0.001

Analytical Test Category III - Extrapolation

Number of Test Cases : 61

Test Category Description : This data set takes some of the more complex shell and plate functions from test categories I and II and examines what happens when the data does not extend to the identical locations in the coordinate axis. For the regular structured grid, the streamwise direction begins at $x = 0.2$ and ends at $x = 0.8$, while the spanwise direction begins at $y = 0.1$ and ends at $y = 0.9$. The irregular grid is shown below. The aerodynamic grid limits begin and end at 0. and 1., respectively.

Test Case Matrix

Grids are defined by streamwise × spanwise elements. A is the amplitude or function value.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
3A	10 × 10 Constant Spac. Plate	10 × 10 Constant Spac. Plate	Linear A=5.0-0.0	Linear A=5.0-0.0
3B			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
3C		20 × 50 Clustered at all Edges Plate	Linear A=5.0-0.0	Constant A=5.0
3D			Linear A=5.0-0.0	Linear A=5.0-0.0
3E			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
3F			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
3G			Linear A=1.01-1.02	Linear A=1.01-1.02
3H			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0
3I	10 × 50 Constant Space Plate	20 × 70 Constant Space Plate	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
3J	50 × 50 Constant Space Plate	70 × 100 Constant Space Plate	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
3K	150 × 150 Constant Space Plate	200 × 200 Constant Space Plate	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02
3A1	10 × 10 Constant Space Shell	10 × 10 Constant Space Shell	Linear A=5.0-0.0	Linear A=5.0-0.0
3B1			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0

3C1		20 × 50 Clustered at all Edges Shell	Linear A=5.0-0.0	Constant A=5.0
3D1			Linear A=5.0-0.0	Linear A=5.0-0.0
3E1			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
3F1			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
3G1			Linear A=1.01-1.02	Linear A=1.01-1.02
3H1			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0
3I1	10 × 50 Constant Space Shell	20 × 70 Constant Space Shell	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
3J1	50 × 50 Constant Space Shell	70 × 100 Constant Space Shell	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
3K1	150 × 150 Constant Space Shell	200 × 200 Constant Space Shell	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02
3A2	10 × 10 Constant Space Plate	10 × 10 Constant Space Shell	Linear A=5.0-0.0	Linear A=5.0-0.0
3B2			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
3C2		20 × 50 Clustered at all Edges Shell	Linear A=5.0-0.0	Constant A=5.0
3D2			Linear A=5.0-0.0	Linear A=5.0-0.0
3E2			Sinusoidal, 1 Cycle, A=2.0	Linear A=5.0-0.0
3F2			Linear A=5.0-0.0	Sinusoidal, 1 Cycle, A=2.0
3G2			Linear A=1.01-1.02	Linear A=1.01-1.02
3H2			Linear A=1.01 - 1.02	Sinusoidal, 1 Cycle, A=2.0
3I2	10 × 50 Constant Space Plate	20 × 70 Constant Space Shell	Sinusoidal, 1 Cycle, A=2.0	Sinusoidal, 3 Cycles, A=2.0
3J2	50 × 50 Constant Space Plate	70 × 100 Constant Space Shell	Sinusoidal, 5 Cycles, A=2.0	Sinusoidal, 3 Cycles, A=0.02
3K2	150 × 150 Constant Space Plate	200 × 200 Constant Space Shell	Sinusoidal, 7 Cycles, A=2.0	Sinusoidal, 7 Cycles, A=0.02

This next set of runs applies the irregular structural grid.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
3A3	Grid 1	10×10 Constant Spac. Plate	Constant A=.5	Constant A=.5
3B3			Constant A=.5	Linear A=.5-0.0
3C3			Linear A=.5-0.0	Constant A=.5
3D3			Linear A=.5-0.0	Linear A=2.5-0.0
3E3			Sinusoidal, 1 Cycle, A=.5	Sinusoidal, 1 Cycle, A=.5
3F3		50×50 Clustered Plate	Constant A=.5	Constant A=.5
3G3			Constant A=.5	Linear A=.5-0.0
3H3			Linear A=.5-0.0	Constant A=.5
3I3			Linear A=.5-0.0	Linear A=2.5-0.0
3J3			Sinusoidal, 1 Cycle, A=.5	Sinusoidal, 1 Cycle, A=.5
3K3			Constant A=0.0025	Constant A=0.0025
3L3			Linear A=0.0025-0.	Linear A=-0.0025-0
3M3			Linear A=0.0025-0	Linear A=0.-0.0025
3N3			Sinusoidal, 1 Cycle, A=0.0005	Sinusoidal, 1 Cycle, A=0.0005
3A4	Grid 1	10×10 Constant Spac. Shell	Constant A=.5	Constant A=.5
3B4			Constant A=.5	Linear A=.5-0.0
3C4			Linear A=.5-0.0	Constant A=.5
3D4			Linear A=.5-0.0	Linear A=2.5-0.0
3E4			Sinusoidal, 1 Cycle, A=.5	Sinusoidal, 1 Cycle, A=.5
3F4		50×50 Clustered Shell	Constant A=.5	Constant A=.5
3G4			Constant A=.5	Linear A=.5-0.0
3H4			Linear A=.5-0.0	Constant A=.5
3I4			Linear A=.5-0.0	Linear A=2.5-0.0
3J4			Sinusoidal, 1 Cycle, A=.5	Sinusoidal, 1 Cycle, A=.5
3K4			Constant A=0.0025	Constant A=0.0025
3L4			Linear A=0.0025-0.	Linear A=-0.0025-0
3M4			Linear A=0.0025-0	Linear A=0.-0.0025
3N4			Sinusoidal, 1 Cycle, A=0.0005	Sinusoidal, 1 Cycle, A=0.0005

Analytical Test Category IV - Diminishing Variation

Number of Test Cases : 30

Test Category Description : A rectangular, Cartesian grid is used for both the structural and aerodynamic grids. Interpolation of various functions are examined: zero-gradient functions (constants), linearly varying functions, sinusoidally varying functions. A series of increasingly fine aerodynamic grids are generated to ensure that the interface method can convert the functional data smoothly without introducing oscillations as the grids become finer.

Test Case Matrix

Grids are defined by streamwise $\times$ spanwise elements. A is the amplitude or function value.

For each case, three finer grids are produced by doubling the number of points in each direction.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
4A	10×10 Constant Spac. Plate	10×10 Constant Spac. Plate	Linear A=2.5-0.0	Linear A=2.5-0.0
4B			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0
4C		20×50 Clustered at all Edges Plate	Linear A=2.5-0.0	Constant A=2.5
4D			Linear A=2.5-0.0	Linear A=2.5-0.0
4E			Sinusoidal, 1 Cycle, A=1.0	Linear A=2.5-0.0
4F			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0
4G			Linear A=.505-.51	Linear A=.505-.51
4H			Linear A=.505 - .51	Sinusoidal, 1 Cycle, A=1.0
4I	10×50 Constant Space Plate	20×70 Constant Space Plate	Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 3 Cycles, A=1.0
4J	50×50 Constant Space Plate	70×100 Constant Space Plate	Sinusoidal, 5 Cycles, A=1.0	Sinusoidal, 3 Cycles, A=0.01
4A1	10×10 Constant Space Shell	10×10 Constant Space Shell	Linear A=2.5-0.0	Linear A=2.5-0.0
4B1			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0

4C1		20 × 50 Clustered at all Edges Shell	Linear A=2.5-0.0	Constant A=2.5
4D1			Linear A=2.5-0.0	Linear A=2.5-0.0
4E1			Sinusoidal, 1 Cycle, A=1.0	Linear A=2.5-0.0
4F1			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0
4G1			Linear A=.505-1.02	Linear A=.505-.51
4H1			Linear A=.505 - 1.02	Sinusoidal, 1 Cycle, A=1.0
4I1	10 × 50 Constant Space Shell	20 × 70 Constant Space Shell	Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 3 Cycles, A=1.0
4J1	50 × 50 Constant Space Shell	70 × 100 Constant Space Shell	Sinusoidal, 5 Cycles, A=1.0	Sinusoidal, 3 Cycles, A=0.01
4A2	10 × 10 Constant Space Plate	10 × 10 Constant Space Shell	Linear A=2.5-0.0	Linear A=2.5-0.0
4B2			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0
4C2		20 × 50 Clustered at all Edges Shell	Linear A=2.5-0.0	Constant A=2.5
4D2			Linear A=2.5-0.0	Linear A=2.5-0.0
4E2			Sinusoidal, 1 Cycle, A=1.0	Linear A=2.5-0.0
4F2			Linear A=2.5-0.0	Sinusoidal, 1 Cycle, A=1.0
4G2			Linear A=.505-.51	Linear A=.505-.51
4H2			Linear A=.505 - .51	Sinusoidal, 1 Cycle, A=1.0
4I2	10 × 50 Constant Space Plate	20 × 70 Constant Space Shell	Sinusoidal, 1 Cycle, A=1.0	Sinusoidal, 3 Cycles, A=1.0
4J2	50 × 50 Constant Space Plate	70 × 100 Constant Space Shell	Sinusoidal, 5 Cycles, A=1.0	Sinusoidal, 3 Cycles, A=0.01

Analytical Test Category V - 1-D Regular Interpolation

Number of Test Cases : 36

Test Category Description : A 1-dimensional, Cartesian grid is used for both the structural and aerodynamic grids. They represent a cantilever beam of unit length. The beam is originally along the x-axis, and every deflection happens in the z direction. Two deflections are considered: a static deflection (bending) due to a distributed force and the first three natural vibration (bending) modes. The first set of cases (test5a-r) has no static deflection, only the three bending modes, with two levels of maximum amplitude (test5a-i, A=1.0; test5j-r, A=0.1), and three different grid sizes (10×10; 100×100; 100×500). The second set of cases (test5a1-r1) has similar structure as the one before, but now there is a superposition of a static deflection (test5a1-i1, A_static = 1.0; test5j1-r1, A_static = 0.1).

Test Case Matrix

Grids are defined by streamwise elements. A is the amplitude or function value related to the bending vibration modes, while A_static is related to a static bending deflection amplitude.

Case Number	Original Grid	Interpolation Grid	Streamwise Function Description	Spanwise Function Description
5a	10 Constant Spacing	10 Constant Spacing	A=1.0 (mode 1) A_static = 0.0	
5b			A=1.0 (mode 2) A_static = 0.0	
5c			A=1.0 (mode 3) A_static = 0.0	
5d	100 Constant Spacing	100 Constant Spacing	A=1.0 (mode 1) A_static = 0.0	
5e			A=1.0 (mode 2) A_static = 0.0	
5f			A=1.0 (mode 3) A_static = 0.0	
5g	100 Constant Spacing	500 Constant Spacing	A=1.0 (mode 1) A_static = 0.0	
5h			A=1.0 (mode 2) A_static = 0.0	
5i			A=1.0 (mode 3) A_static = 0.0	
5j	10 Constant Spacing	10 Constant Spacing	A=0.1 (mode 1) A_static = 0.0	
5k			A=0.1 (mode 2) A_static = 0.0	
5l			A=0.1 (mode 3) A_static = 0.0	

5m	100 Constant Spacing	100 Constant Spacing	A=0.1 (mode 1) A_static = 0.0	
5n			A=0.1 (mode 2) A_static = 0.0	
5o			A=0.1 (mode 3) A_static = 0.0	
5p	100 Constant Spacing	500 Constant Spacing	A=0.1 (mode 1) A_static = 0.0	
5q			A=0.1 (mode 2) A_static = 0.0	
5r			A=0.1 (mode 3) A_static = 0.0	
5a1	10 Constant Spacing	10 Constant Spacing	A=1.0 (mode 1) A_static = 1.0	
5b1			A=1.0 (mode 2) A_static = 1.0	
5c1			A=1.0 (mode 3) A_static = 1.0	
5d1	100 Constant Spacing	100 Constant Spacing	A=1.0 (mode 1) A_static = 1.0	
5e1			A=1.0 (mode 2) A_static = 1.0	
5f1			A=1.0 (mode 3) A_static = 1.0	
5g1	100 Constant Spacing	500 Constant Spacing	A=1.0 (mode 1) A_static = 1.0	
5h1			A=1.0 (mode 2) A_static = 1.0	
5i1			A=1.0 (mode 3) A_static = 1.0	
5j1	10 Constant Spacing	10 Constant Spacing	A=0.1 (mode 3) A_static = 0.1	
5k1			A=0.1 (mode 3) A_static = 0.1	
5l1			A=0.1 (mode 3) A_static = 0.1	
5m1	100 Constant Spacing	100 Constant Spacing	A=0.1 (mode 3) A_static = 0.1	
5n1			A=0.1 (mode 3) A_static = 0.1	
5o1			A=0.1 (mode 3) A_static = 0.1	
5p1	100 Constant Spacing	500 Constant Spacing	A=0.1 (mode 3) A_static = 0.1	

5q1			A=0.1 (mode 3) A_static = 0.1	
5r1			A=0.1 (mode 3) A_static = 0.1	

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX C - ALGORITHM CODES

This appendix includes the software necessary to reproduce the test cases in this report. In order to run the NUBS option, it is necessary to obtain a DT_NURBS SPLINE GEOMETRY SUBROUTINE LIBRARY license. This license can be obtained from Robert Ames, NSWC/Carderock Division, ames@oasys.dt.navy.mil. There are restrictions on the source files of this library.

```
csdcfd.par:
c user-defined parameters
c      imxs = streamwise (x-direction) points on structural grid
c      jmxs = spanwise (y-direction) points on structural grid
c      kmxs = normal (z-direction) points on structural grid
c         *** kmxs=1 --- always!
c      imxa = streamwise (x-direction) points on aerodynamic grid
c      jmxa = spanwise (y-direction) points on aerodynamic grid
c      kmxa = normal (z-direction) points on aerodynamic grid
c         *** kmxa=1 --- always!
c      nwk = work array for nubs (dt_nurbs)
c            30k appears to be fine for all cases examined
c            DT_NURBS will flag if it's not high enough
       parameter (imxs=20,jmxs=30,kmxs=1)
       parameter (imxa=220,jmxa=80,kmxa=1)
       parameter (nwk=30000)
c  computed parameters
c      dnstr = total number of structural data points on surface
c      dnaro = total number of aerodynamic data points on surface
       integer dnstr,dnaro
       parameter (dnaro=imxa*jmxa*kmxa,dnstr=imxs*jmxs*kmxs)
c
**********************************************************************
c
c   This program handles the 4 3-D methods
c

        program main
        implicit double precision (a-h,o-z)
        include 'csdcfd.par'


        external cputim
c
c      xs,ys,zs     - structural grid points
c      sxs,sys,szs - structural mode shapes values at grid points
c      xa,yz,za     - aerodynamic grid points
c      sxa,sya,sza - calculated aerodynamic deflections
c      isg,jsg,ksg - number of structural points
c      is,js,ks     - number of structural points ( for each mode)
c      ia,ja,ka     - number of aerodynamic points
c
c
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
       common /str/  xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       common/filen/fnm
```

```
         common /struct/ sxs(imxs,jmxs,kmxs)
      &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
      &,js,ks,isg,jsg,ksg
        dimension xs(imxs,jmxs,kmxs),ys(imxs,jmxs,kmxs),
      1  zs(imxs,jmxs,kmxs)

         dimension f(dnaro,3)
c --- device dependent        real oldtim,time,cputim
         character*80 fnm,fnm1,fnm2,fnm3,fnm4
c
c  Read in the file names containing the input data
c
111      continue
c
c  Clear all of the variables
c
         npts=0
         nspts=0
         do l=1,dnaro
          xa(l)=0.0
          ya(l)=0.0
          za(l)=0.0
          fa(l)=0.0
          f(l,3)=0.0
         enddo
         do l=1,dnstr
          xt(l)=0.0
          yt(l)=0.0
          zt(l)=0.0
          ft(l)=0.0
         enddo
         ft(dnstr+1)=0.0
         ft(dnstr+2)=0.0
         ft(dnstr+3)=0.0
c
c  Read in the new filename information
c
c*** -- device dependent        oldtim=0.0
         write(*,*)'Enter the structural grid filename'
         read(*,'(A)',end=9000)fnm1
         write(*,*)'Enter the structural data filename'
         read(*,'(A)')fnm2
         write(*,*)'Enter the aerodynamic grid filename'
         read(*,'(A)')fnm3

     open(unit=13,file=fnm1,status='unknown')
     open(unit=15,file=fnm2,status='unknown')
     open(unit=17,file=fnm3,status='unknown')

c
c read in data pertaining to structural grid and data
c

     read(13,*)isg,jsg,ksg
```

```
      if(isg.eq.0) isg=1
      if(jsg.eq.0) jsg=1
      if(ksg.eq.0) ksg=1
      nspts=isg*jsg*ksg
c
c         Error check on dimensions
c
      if(nspts.gt.dnstr) then
         write(6,*) ' You have exceeded the dnstr dimension ',
     1    nspts
         stop
      endif
      if(isg.gt.imxs) then
         write(6,*) ' You have exceeded the imxs dimension ',
     1    isg
         stop
      endif
      if(jsg.gt.jmxs) then
         write(6,*) ' You have exceeded the jmxs dimension ',
     1    jsg
         stop
      endif
      if(ksg.gt.kmxs) then
         write(6,*) ' You have exceeded the kmxs dimension ',
     1    ksg
         stop
      endif
c            End error checking
c
c            Read remainder of structural file
c
      read(13,*) (xt(m),m=1,nspts),
     &   (yt(m),m=1,nspts),(zt(m),m=1,nspts)
      close(13)

   read(15,*)ns
      read(15,*)is,js,ks
      if(is.eq.0) is=1
      if(js.eq.0) js=1
      if(ks.eq.0) ks=1
      if(is.ne.isg.or.js.ne.jsg.or.ks.ne.ksg) then
         write(6,*) ' Grid and Data Dimensions Dont Match'
         write(6,*) ' I : ',isg,is
         write(6,*) ' J : ',jsg,js
         write(6,*) ' K : ',ksg,ks
         stop
       endif
    read(15,*)
     &   (((sxs(i,j,k),i=1,is),j=1,js),k=1,ks),
     &   (((sys(i,j,k),i=1,is),j=1,js),k=1,ks),
     &   (((szs(i,j,k),i=1,is),j=1,js),k=1,ks)
       close(15)
c
c read in data pertaining to aerodynamic grid and data
```

```
c
    read(17,*)ia,ja,ka
         if(ia.eq.0) ia=1
         if(ja.eq.0) ja=1
         if(ka.eq.0) ka=1
         npts=ia*ja*ka
c
c               Error check on dimensions
c
         if(npts.gt.dnaro) then
            write(6,*) ' You have exceeded the dnaro dimension ',
     1       npts
            stop
         endif
         if(ia.gt.imxa) then
            write(6,*) ' You have exceeded the imxa dimension ',
     1       ia
            stop
         endif
         if(ja.gt.jmxa) then
            write(6,*) ' You have exceeded the jmxa dimension ',
     1       ja
            stop
         endif
         if(ka.gt.kmxa) then
            write(6,*) ' You have exceeded the kmxa dimension ',
     1       ka
            stop
         endif
c               End error checking
c
c               Finish reading aerodynamic data
           read(17,*) (xa(m),m=1,npts),(ya(m),m=1,npts),
     &     (za(m),m=1,npts)
         close(17)
c
c             fill output arrays with zeros
c
         do m=1,3
         do l=1,npts
         f(l,m)=0.0
         enddo
         enddo
c
c Prompt user for which scheme to use for interpolation
c
 112  continue
      write(*,1)
      write(*,2)
      write(*,4)
      write(*,5)
      write(*,6)
      read(*,*) meth
```

```
c         assume that only one mode shape per file!
      mode=1
      if(meth.lt.0.or.meth.gt.4) go to 112
c
c Call subroutine of chosen method
c
        if(meth .eq. 1) then
c***--device dependent        stime=cputim(oldtim)
c           Infinite Plate Spline
c               (Must choose what components here)
c
          write(*,7)
          read(*,*) ncomp
          if(ncomp.lt.4) then
             nss=ncomp
             nee=ncomp
          else
             nss=1
             nee=3
          endif
          do l=nss,nee
          call load(mode,l,nspts,ft)
          call ips(mode,ierr)
          do m=1,npts
            f(m,l)=fa(m)
          enddo
c             redefine aerodynamic data (modified in ips)
      open(unit=17,file=fnm3,status='unknown')
      read(17,*)ia,ja,ka
          read(17,*) (xa(m),m=1,npts),(ya(m),m=1,npts),
     &    (za(m),m=1,npts)
        close(17)
          enddo
        elseif(meth .eq. 2)then
c                     Thin Plate Spline
          call mq(3,f)
        elseif(meth .eq. 3)then
c                     Multiquadrics
          call mq(1,f)
        elseif(meth .eq. 4)then
c                     NUBS
          call nurbs(mode,f)
        endif
c
c  Check amount of required cpu runtime for this method
c
c*** -- device dependent      time=cputim(stime)

      call output(mode,ia,ja,ka,f)

      close(13)
      close(15)
      close(17)
```

```
c*** -- device dependent       time=abs(time-stime)
       call calcmax(mode,f)
c*** -- device dependent       oldtime=time
       go to 111
9000   continue
c
c           format statements
c
 1     format(///4x,'Enter choice of interpolation methods:')
 2     format(7x,'1]  Infinite Plate Splines')
 4     format(7x,'2]  Thin Plate Spline')
 5     format(7x,'3]  Multi-Quadrics')
 6     format(7x,'4]  NUBS')
 7     format(///4x,
     1' What kind of data is to be interpolated?',/,
     17x,' 1]  x-component only ',/,
     27x,' 2]  y-component only ',/,
     37x,' 3]  z-component only ',/,
     47x,' 4]  all three components ')
c
      stop
      end
c
c           This subroutine generates the output files
c
      subroutine output(mode,ia,ja,ka,f)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts

       dimension f(dnaro,3)
       common/filen/fnm

      character*80  ofile,nfile,mfile,lfile,fnm

       write(6,*) ' Enter the output file type : '
       write(6,*) ' 0 : Plot3d , 1 : SGF '
       read(*,*) itype
       write(6,*)' Enter aerodynamic grid + mode shape file name'
       read(*,'(A)')nfile
       write(6,*)' Enter interpolated function file name'
       write(6,*)' Must end in character x '
       read(*,'(A)')ofile
 10     continue

       nvar=1
      if(itype.eq.0) then
       open(unit=102,file=nfile,status='unknown')
       write(102,*)ia,ja,ka
       write(102,*)((xa(m)+f(m,1)),m=1,npts),
     &              ((ya(m)+f(m,2)),m=1,npts),
     &              ((za(m)+f(m,3)),m=1,npts)
       write(103,*)ia,ja,ka
       write(103,*)((f(m,1)),m=1,npts),
```

```
     &              ((f(m,2)),m=1,npts),
     &              ((f(m,3)),m=1,npts)
      close(102)
      close(103)
      go to 1000
c
      open(unit=103,file=ofile,status='unknown')
      write(103,*)ia,ja,ka,nvar
      write(103,*)(f(ll,1),ll=1,npts)
      close(103)
c
      do ll=1,80
        if(ofile(ll:ll).eq.'x') ie=ll
      enddo
      ofile(1:ie)=ofile(1:ie-1)//'y'
      open(unit=103,file=ofile,status='unknown')
      write(103,*)ia,ja,ka,nvar
      write(103,*)(f(ll,2),ll=1,npts)
      close(103)
c
      do ll=1,80
        if(ofile(ll:ll).eq.'y') ie=ll
      enddo
      ofile(1:ie)=ofile(1:ie-1)//'z'
      open(unit=103,file=ofile,status='unknown')
      write(103,*)ia,ja,ka,nvar
      write(103,*)(f(ll,3),ll=1,npts)
      close(103)
     else
      open(unit=102,file=nfile,status='unknown')
      nz=1
      nr=0
      write(102,'(A)') nfile
      write(102,*) nz
      write(102,*)ia,ja,ka
      write(102,'(A)') nfile
      write(102,*) nr
      do k=1,ka
      do j=1,ja
      iss=(j-1)*ia+1
      iee=j*ia
      write(102,*)(xa(m)+f(m,1),m=iss,iee)
      write(102,*)(ya(m)+f(m,2),m=iss,iee)
      write(102,*)(za(m)+f(m,3),m=iss,iee)
      enddo
      enddo
      close(102)
      do l=1,3
      if(l.eq.2) then
         do ll=1,80
           if(ofile(ll:ll).eq.'x') ie=ll
         enddo
         ofile(1:ie)=ofile(1:ie-1)//'y'
      else if (l.gt.2) then
```

```
        do ll=1,80
          if(ofile(ll:ll).eq.'y') ie=ll
        enddo
        ofile(1:ie)=ofile(1:ie-1)//'z'
      endif
      open(unit=103,file=ofile,status='unknown')
      do k=1,ka
      do j=1,ja
      iss=(j-1)*ia+1
      iee=j*ia
      write(103,*)(f(m,1),m=iss,iee)
      enddo
      enddo
      close(103)
      enddo
      endif
c
1000  continue
      return
      end
c
c         This subroutine calculates the max/min locations and
c               magnitudes
c         (From MPROC2D)
      subroutine calcmax(mode,f)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'

c
c      xs,ys,zs     - structural grid points
c      sxs,sys,szs - structural mode shapes values at grid points
c      xa,yz,za     - aerodynamic grid points
c      sxa,sya,sza - calculated aerodynamic deflections
c      isg,jsg,ksg - number of structural points
c      is,js,ks     - number of structural points ( for each mode)
c      ia,ja,ka     - number of aerodynamic points
c
c
        common /struct/ sxs(imxs,jmxs,kmxs)
     1,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     1,js,ks,isg,jsg,ksg
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
       common /str/  xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       dimension f(dnaro,3)
C
C  Add to the existing file 7 the check data!
    open(unit=7,file='accuracy.dat',status='unknown')
c  141   read(7,*,end=142)
c        go to 141
c  142   continue
C First, find maximum in structural data
        tmp = dsqrt(sxs(1,1,1)*sxs(1,1,1)
     1  + sys(1,1,1)*sys(1,1,1)
     1  + szs(1,1,1)*szs(1,1,1))
```

```
        tmp2 = tmp
        tmp3 = 0
        locmax=1
        locmin=1
        m=0
        do k=1,ksg
        do j=1,jsg
        do i=1,isg
        m=m+1
        chktmp =dsqrt(sxs(i,j,k)*sxs(i,j,k)
     1   +sys(i,j,k)*sys(i,j,k)
     1   +szs(i,j,k)*szs(i,j,k))
        tmp3 = tmp3 + chktmp
        if (chktmp.gt.tmp) then
           tmp = chktmp
           locmax = m
        endif
        if(chktmp.lt.tmp2) then
           tmp2 = chktmp
           locmin =m
        endif
      enddo
      enddo
      enddo
      pmax = tmp
      pmin = tmp2
      pavg = tmp3/nspts

C Find maximum values in interpolated data
        tmp = dsqrt(f(1,1)*f(1,1)+f(1,2)*f(1,2)+f(1,3)*f(1,3))
        tmp2 = tmp
        tmp3 = 0.0
        min=0
        max=0

        do 10 m=1,npts
           chktmp = dsqrt(f(m,1)*f(m,1)+f(m,2)*f(m,2)
     >          +f(m,3)*f(m,3))
           tmp3 = tmp3 + chktmp
           if (chktmp.gt.tmp) then
              tmp = chktmp
              max = m
           endif
           if(chktmp.lt.tmp2) then
              tmp2 = chktmp
              min = m
           endif
10      continue
        dmax = tmp
        dmin = tmp2
        davg = tmp3/npts

C Print max values
     if (pmax.eq.0.0) then
```

```
            pdiff = 100*(dmax-pmax)
         else
            pdiff = 100*(dmax-pmax)/pmax
         endif

         write(6,8) pmax,xt(locmax),yt(locmax),zt(locmax)
         write(6,9) dmax,xa(max),ya(max),za(max)
         write(6,13) dmax-pmax,pdiff
         write(7,8) pmax,xt(locmax),yt(locmax),zt(locmax)
         write(7,9) dmax,xa(max),ya(max),za(max)
         write(7,13) dmax-pmax,pdiff

C Print min values
     if (pmin.eq.0.0) then
            pdiff = 100*(dmin-pmin)
         else
            pdiff = 100*(dmin-pmin)/pmin
         endif
         write(6,11) pmin,xt(locmin),yt(locmin),zt(locmin)
         write(6,12) dmin,xa(min),ya(min),za(min)
         write(6,13) dmin-pmin,pdiff
         write(7,11) pmin,xt(locmin),yt(locmin),zt(locmin)
         write(7,12) dmin,xa(min),ya(min),za(min)
         write(7,13) dmin-pmin,pdiff
    8    format( ' The true maximum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
    9    format( ' The interpolated maximum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
   11    format( ' The true minimum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
   12    format( ' The interpolated minimum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
   13    format( ' The difference is ',g13.5,'(',g9.3,'%)')
      return
      end
      subroutine ips(mode,ierr)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c    This subroutine solves the infinite plate spline equations
c       for a three-dimensional surface
c    The equations in this code were extracted from a code developed
c       by Ram Shan at GTRI in 1993 named MPROC3D. This code was a
c       3-D extension of a NASA-Langley code called MPROC, provided
c       by Beth Rausch in the Unsteady Aerodynamics Branch.
c
c
c    The code assumes that each mode is called independently for
c       each direction (up to 3 directions).
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c
      dimension singular(3),nd(3),emax(3)
```

```
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
      common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
      common /surf/ xb(dnstr),yb(dnstr)
      common /work/ array(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), dum(dnstr+3)

c
      data eps/1.e-28/
      data singular/0.,0.,0./

      do l=1,dnstr+3
       w(l)=0.0
       xb(l)=0.0
       yb(l)=0.0
       dum(l)=0.0
       do ll=1,dnstr+3
         array(l,ll)=0.0
         v(l,ll)=0.0
       enddo
      enddo


c      singular(1)  = 5.0e-4
c      singular(2)  = 5.0e-4
c      singular(3)  = 1.0e-4
c      singular(1)  = 1.0e-3
c      singular(2)  = 1.0e-3
c      singular(3)  = 5.0e-4

c     ---------------------------------------------------
c     check to see if surface is planar
c     ---------------------------------------------------
c
      call plane(ipdir,itmp)
      if(ipdir.lt.1.or.ipdir.gt.3) then
          write(6,*) ' Error in plane subroutine '
          write(6,*) ' Plane returned is ',ipdir
          stop
      endif

c     ---------------------------------------------------
c     ipdir = 1 x-y plane splinal matrix data
c     ipdir = 2 x-z plane splinal matrix data
c     ipdir = 3 y-z plane splinal matrix data
c     ---------------------------------------------------
      if (itmp.eq.1) then
     if (ipdir.eq.1) then
c           ipdir=1 is the x-y plane, data to update is z (i2=3)
        i2s = 3
        i2e = 3
         endif
     if (ipdir.eq.2) then
c           ipdir=2 is the x-z plane, data to update is y (i2=2)
        i2s = 2
```

```
         i2e = 2
           endif
     if (ipdir.eq.3) then
c            ipdir=3 is the y-z plane, data to update is x (i2=1)
         i2s = 1
         i2e = 1
           endif
       else
     i2s = 1
     i2e = 3
       endif


       rbu1 = 0.0
       rbu2 = 0.0
       rbu3 = 0.0
       rbv1 = 0.0
       rbv2 = 0.0
       rbv3 = 0.0

       do 5 i2=i2s,i2e
c
c store appropriate values in x & y arrays in order to use
c existing subroutines
c
       if (i2 .eq. 1) then
     write(6,*) 'Processing x-y coordinate data'
          call fdinert(xt,yt,nspts,thz,rbu3, rbv3,xcg,ycg)
          do ii=1,nspts                      ! use x & y coordinates
             xb(ii) = (xt(ii) - xcg)/rbu3      ! shift coords to cg
             yb(ii) = (yt(ii) - ycg)/rbv3
c
c  Experienced division by zero read: beta=atan2(yb(ii),xb(ii)
c
             beta = atan2(yb(ii), xb(ii)+eps)
             rad = sqrt(xb(ii)**2 + yb(ii)**2)  ! rotate to principle axes
        xb(ii) = rad*cos(beta-thz)
        yb(ii) = rad*sin(beta-thz)
          enddo
       endif
       if (i2 .eq. 2) then
          write(6,*) 'Processing x-z coordinate data'
          call fdinert(xt,zt,nspts,thy,rbu2,rbv2,xcg,zcg)
          do 29 ii=1,nspts                    ! use x & z coordinates
             xb(ii) = (xt(ii) - xcg)/rbu2      ! shift coords to cg
             yb(ii) = (zt(ii) - zcg)/rbv2
c
c  Incase of Division by zero beta was beta=atan2(yb(ii),xb(ii))
c
        beta = atan2(yb(ii), xb(ii)+eps)
             rad = sqrt(xb(ii)**2 + yb(ii)**2)  ! rotate to principle axes
        xb(ii) = rad*cos(beta-thy)
        yb(ii) = rad*sin(beta-thy)
29        continue
```

```
      endif
      if (i2 .eq. 3) then
     write(6,*) 'Processing y-z coordinate data'
        call fdinert(yt,zt,nspts,thx,rbu1,rbv1,ycg,zcg)
        do 30 ii=1,nspts                      ! use y & z coordinates
          xb(ii) = (yt(ii) - ycg)/rbu1         ! shift coords to cg
          yb(ii) = (zt(ii) - zcg)/rbv1
c
c  Division by zero beta was : beta=atan2(yb(ii),xb(ii))
c
       beta = atan2(yb(ii), xb(ii)+eps)
          rad = sqrt(xb(ii)**2 + yb(ii)**2)  ! rotate to principle axes
       xb(ii) = rad*cos(beta-thx)
       yb(ii) = rad*sin(beta-thx)
30      continue
      endif
c
c           fill solution matrix array
c
      do 2 i=1,nspts
        do 1 j=1,i
          r2 = (xb(i)-xb(j))**2+(yb(i)-yb(j))**2
          g= r2 * dlog(r2+eps)
       array(i,j) = g
       array(j,i) = g
    1   continue
    2 continue

      do 3 j=1,nspts
     array(nspts+1,j) = 1.0
     array(j,nspts+1) = 1.0
    3 continue

      array(nspts+1,nspts+1) = 0.0

      do 4 j=1,nspts
     array(nspts+2,j) = xb(j)
     array(j,nspts+2) = xb(j)

    4 continue

      array(nspts+2, nspts+1) = 0.0
      array(nspts+1, nspts+2) = 0.0
      array(nspts+2, nspts+2) = 0.0

      do 6 j=1,nspts
     array(nspts+3,j) = yb(j)
     array(j,nspts+3) = yb(j)
    6 continue

      array(nspts+3, nspts+1) = 0.0
      array(nspts+1, nspts+3) = 0.0
      array(nspts+3, nspts+2) = 0.0
      array(nspts+2, nspts+3) = 0.0
```

```
      array(nspts+3, nspts+3) = 0.0

      n1=nspts+3
c
c      ---------------------------------------------------------
c      call routine to decompose main matrix.
c      ---------------------------------------------------------
       call svdcmp(n1)
c
c scale structural coordinates & cfd grid coordinates
c rotate coordinates to principle axes
c Select appropriate normal coordinate for interpolation
c
          if (i2.eq.1) then
             write(6,*) 'Calculating xy-plane deflections'
             do m=1,npts
                xa(m) = (xa(m) - xcg)/rbu3
                ya(m) = (ya(m) - ycg)/rbv3
       rad = sqrt(xa(m)**2 + ya(m)**2)
       beta = atan2(ya(m),xa(m)+eps)
                xa(m) = rad*cos(beta-thz)
       ya(m) = rad*sin(beta-thz)
             enddo
          elseif (i2.eq.3) then
             write(6,*) 'Calculating yz-plane deflections'
             do m=1,npts
                ya(m) = (ya(m) - ycg)/rbu1
                za(m) = (za(m) - zcg)/rbv1
       beta = atan2(za(m),ya(m)+eps)
       rad = sqrt(ya(m)**2 + za(m)**2)
       ya(m) = rad*cos(beta-thx)
       za(m) = rad*sin(beta-thx)
             enddo
          else if (i2.eq.2) then
             write(6,*) 'Calculating xz-plane deflections'
             do m=1,npts
                xa(m) = (xa(m) - xcg)/rbu2
               za(m) = (za(m) - zcg)/rbv2
       beta = atan2(za(m), xa(m)+eps)
       rad = sqrt(xa(m)**2 + za(m)**2)
       xa(m) = rad*cos(beta-thy)
       za(m) = rad*sin(beta-thy)
             enddo
          else
             write(6,*) ' Code error no plane identified ',i2
       endif
c
       ft(nspts+1) = 0.
       ft(nspts+2) = 0.
       ft(nspts+3) = 0.
c
c  Ask if user wants to change the threshold
c
       irest=0
```

```
      call chthres(nspts,irest,ipdir,singular(ipdir),nd(ipdir),
     1     emax(ipdir))
      nn=nspts+3
      nnn=dnstr+3
      call slvdriver(nn,nnn,nd(ipdir),singular(ipdir),
     1     emax(ipdir))

c
c     -------------------------------------------
c     compute deflections at collocation points
c     -------------------------------------------

      if(i2.eq.1) then
c            update z deflections
      do m=1,npts
        fa(m)=zfun(xa(m),ya(m),eps)
      enddo
      else if (i2.eq.2) then
c            update y deflections
      do m=1,npts
        fa(m)=zfun(xa(m),za(m),eps)
      enddo
      else if (i2.eq.3) then
c            update x deflections
      do m=1,npts
        fa(m)= zfun(ya(m),za(m),eps)
      enddo
      endif
5     continue
c
      return
      end
c
c     FFFFFFFFFFFFFFFFFFFF
      function zfun(xx,yy,eps)
c     FFFFFFFFFFFFFFFFFFFF
      implicit double precision (a-h,o-z)
     include 'csdcfd.par'
     common /str/ xt(dnstr),yt(dnstr),zt(dnstr),zeta(dnstr+3),nspts
     common /surf/ xi(dnstr),eta(dnstr)
c     ------------------------------------------------------------
c     spline evaluation function for a symmetric surface spline
c     ------------------------------------------------------------
      b1=zeta(nspts+1)
      b2=zeta(nspts+2)
      b3=zeta(nspts+3)
      zfun = b1+b2*xx+b3*yy

      do 1 i=1,nspts
         dx2 = (xx-xi(i))**2
         dy2 = (yy-eta(i))**2
         rl = dx2 + dy2
     zfun = zfun + zeta(i)*rl*dlog(rl+eps)
   1  continue
```

```
        return
        end
c-----------------------------------------------------------------
      subroutine slvdriver(n3,mp,nd,singular,emax)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),b(dnstr+3),nspts
      common /work/ u(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), x(dnstr+3)
c
      common /file/ iunit
c
c
c Edit diagonal

      nd = 0.0
      wmax = 0.0
      do i=1, n3
     wmax = max(wmax, w(i))
      enddo
      emax = wmax
      wmin=wmax*singular
      do i=1, n3
    if (w(i) .lt. wmin) then
       w(i) = 0.0
       nd = nd + 1
    endif
      enddo
c
c Call Solver
c
      call svbksb(n3, n3 )
c
c Copy solution matrix into b vector
c
      do i = 1, n3
         b(i) = x(i)
      enddo

      return
      end
C -------------------------------------------------------------------
      subroutine svbksb(m,n)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),b(dnstr+3),nspts
      common /work/ u(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), x(dnstr+3)
      dimension tmp(dnstr+3)
C -------------------------------------------------------------------
C This subroutine solves Ax = b where A = U, W, V as returned
C from svdcmp
C m,n = logical dimensions of A
```

```
C b = RHS vector
C x = output solution vector
C ---------------------------------------------------------------------
      if (n.ne.m) n=m
      do j=1,n
         s=0.0
         if (w(j).ne.0.0) then
            do i=1,m
               s=s+u(i,j)*b(i)
            enddo
            s=s/w(j)
         endif
         tmp(j) = s
      enddo
      do j=1,n
         s=0.0
         do jj=1,n
            s=s+v(j,jj)*tmp(jj)
         enddo
         x(j) = s
      enddo
      return
      end
c-------------------------------------------------
      subroutine fdinert(xf,yf,n, th, rbu, rbv, xcg, ycg)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c      dimension xf(n), yf(n)
      dimension xf(dnstr), yf(dnstr)
c --------------------------------------------------------------------
c This subroutine finds the moments of inertia for given coordinate info
c The moments of inertia is used to find the principle axes direction
c It assumes that each coordinate represents a particle of mass = 1
c xf, yf - coordinate info
c th - principle axes direction angle (rad)
c rbu - radius of gyration in u-direction
c rbv - radius of gyration in v-direction
c --------------------------------------------------------------------
c
      xi = 0.0
      yi = 0.0
      xyi = 0.0
      qx = 0.0
      qy = 0.0
c
c Calculate 1st & 2nd moments of inertia
c
      do ii= 1, n
         xi = xi + yf(ii)*yf(ii)
         yi = yi + xf(ii)*xf(ii)
         xyi = xyi + xf(ii)*yf(ii)
         qx = qx + yf(ii)
         qy = qy + xf(ii)
      enddo
```

```
c
c Calculate cg locations
c
      xcg = qy / n
      ycg = qx / n
c
c Calculate direction of principle axes & principle moments
c
      tmp = xi - yi
      if (tmp.lt.0.0) then
         th  = .5*atan2(-1.0*xyi, .5*tmp)
         th1 = .5*atan2(xyi, -.5*tmp)
      else
         th  = .5*atan2(-1.0*xyi, .5*tmp)
         th1 = .5*atan2(xyi, -.5*tmp)
      endif
      ui = xi*(cos(th))**2 -2.0*xyi*sin(th)*cos(th) + yi*(sin(th))**2
      vi = xi*(cos(th1))**2 -2.0*xyi*sin(th1)*cos(th1)+yi*(sin(th1))**2
c
c Calculate radii of gyration
c
      rbu = sqrt(ui/n)
      rbv = sqrt(vi/n)

      return
      end
*-----------------------------------------------------------------
      subroutine svdcmp(m)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /work/ a(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), dum(dnstr+3)
      dimension rv1(dnstr+3)
*-----------------------------------------------------------------
* This subroutine performs a Singular Value Decomposition (SVD)
* a = u * w * (v)-transpose
* It is obtained from
*   "Numerical Recipes", By  Press & Flannery et al.,
*      Cambridge University Press, 1989, pgs. 57- 64
* a = matrix logical dimensions m by n
*     physical dimensions mp by np
* u = stored in a on output
* w = diagonal matrix of singular values
* v = NOT stored as transpose on output
* if n > m, fill A to make square with zero rows
*-----------------------------------------------------------------

      n=m
c
c
      if (m.lt.n) then
     write(6,*) 'Error:Inside SVDCMP routine'
     write(6,*) 'Augment A matrix with zero rows'
     stop
```

```
      endif
*
* Householder reduction to bidiagonal form
*
      g = 0.0
      scale = 0.0
      anorm = 0.0
      do 25 i=1,n
     l=i+1
     rv1(i)=scale*g
     g=0.0
     s=0.0
          scale = 0.0
     if (i.le.m) then
        do k=i,m
           scale=scale+abs(a(k,i))
            enddo
        if (scale.ne.0.0) then
           do k=i,m
          a(k,i) = a(k,i)/scale
          s=s+a(k,i)*a(k,i)
               enddo
           f=a(i,i)
           g=-sign(sqrt(s),f)
           h=f*g-s
           a(i,i)=f-g
           if (i.ne.n) then
          do j=l,n
             s=0.0
             do k=i,m
            s=s+a(k,i)*a(k,j)
                      enddo
             f=s/h
             do k=i,m
            a(k,j) = a(k,j)+f*a(k,i)
                      enddo
                   enddo
               endif
           do k=i,m
          a(k,i) = scale*a(k,i)
               enddo
            endif
         endif
     w(i) = scale * g
     g=0.0
     s=0.0
     scale=0.0
     if((i.le.m).and.(i.ne.n)) then
        do k=l,m
           scale=scale+abs(a(i,k))
            enddo
        if (scale.ne.0.0) then
```

```
          do k=1,n
         a(i,k) = a(i,k)/scale
         s=s+a(i,k)*a(i,k)
          enddo
          f=a(i,l)
          g=-sign(sqrt(s),f)
          h=f*g-s
          a(i,l) = f - g
          do k=1,n
        rv1(k) = a(i,k)/h
          enddo
          if(i.ne.m) then
         do j=l,m
            s=0.0
            do k=l,n
           s=s+a(j,k)*a(i,k)
            enddo
            do k=l,n
           a(j,k) = a(j,k)+s*rv1(k)
            enddo
         enddo
          endif
          do k=l,n
         a(i,k) = scale*a(i,k)
          enddo
        endif
         endif
     anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25    continue
*
* Accumulation of right hand side transformations
*
      do 32 i=n,1,-1
         if (i.lt.n) then
            if (g.ne.0.0) then
                do j=l,n
                   v(j,i)=(a(i,j)/a(i,l))/g
                 enddo
            do j=l,n
               s=0.0
               do k=l,n
                  s=s+a(i,k)*v(k,j)
               enddo
               do k=l,n
                  v(k,j)=v(k,j)+s*v(k,i)
               enddo
            enddo
            endif
            do j=l,n
               v(i,j) = 0.0
               v(j,i) = 0.0
            enddo
         endif
         v(i,i) = 1.0
```

```
         g = rv1(i)
         l = i
32    continue
*
* Accumulation of Left Hand Side transformations
*
      do 39 i=n,1,-1
         l=i+1
         g=w(i)
         if (i.lt.n) then
            do j = l,n
              a(i,j)=0.0
            enddo
         endif
         if (g.ne.0.0) then
            g = 1.0/g
            if (i.ne.n) then
               do j=l,n
                  s=0.0
                  do k=l,m
                     s=s+a(k,i)*a(k,j)
                  enddo
                  f=(s/a(i,i))*g
                  do k=i,m
                     a(k,j)=a(k,j)+f*a(k,i)
                  enddo
               enddo
            endif
            do j=i,m
               a(j,i)=a(j,i)*g
            enddo
         else
            do j=i,m
               a(j,i) = 0.0
            enddo
         endif
         a(i,i) = a(i,i) + 1.0
39    continue
*
* Diagonalization of bidiagonal form
*
      do 49 k=n,1,-1        ! singular values loop
      do 48 its=1,30      ! iteration loop

         do l=k,1,-1      ! test for splitting
            nm=l-1        ! rv1(1) always zero
            if ((abs(rv1(l))+anorm).eq.anorm) goto 2
            if ((abs(w(nm))+anorm).eq.anorm) goto 1
         enddo
1        c=0.0            ! cancellation of rv1(l), if l>1
         s=1.0
         do i=l,k
            f = s*rv1(i)
            rv1(i) = c*rv1(i)
```

```
            if ((abs(f)+anorm).eq.anorm) goto 2
            g = w(i)
            h = sqrt(f*f+g*g)
            w(i) = h
            h = 1.0/h
            c = (g*h)
            s = -(f*h)
            do j=1,m
               y=a(j,nm)
               z=a(j,i)
               a(j,nm) = (y*c) + (z*s)
               a(j,i)  = -(y*s) + (z*c)
            enddo
         enddo
2        z = w(k)
         if (l.eq.k) then            ! Convergence
            if (z.lt.0.0) then       ! singular value made > 0
               w(k) = -z
               do j=1,n
                 v(j,k) = -v(j,k)
               enddo
            endif
            goto 3
         endif
         if (its.eq.30) then
            write(6,*) 'Error: In SVDCMP routine'
            write(6,*) 'No convergence in 30 iterations'
            write(6,*) 'Matrix cannot be inverted!'
            stop
         endif
         x = w(l)
         nm = k-1
         y = w(nm)
         g = rv1(nm)
         h = rv1(k)
         f = ((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y)
         g = sqrt(f*f  + 1.0)
         f = ((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x

c Next QR transformation

         c = 1.0
         s = 1.0
         do j=l,nm
            i=j+1
            g=rv1(i)
            y=w(i)
            h=s*g
            g=c*g
            z=sqrt(f*f+h*h)
            rv1(j) = z
            c = f/z
            s = h/z
            f = (x*c)+(g*s)
```

```
            g = -(x*s) + (g*c)
            h = y*s
            y = y*c
            do jj=1,n
                x = v(jj,j)
                z = v(jj,i)
                v(jj,j) = (x*c)+(z*s)
                v(jj,i) = -(x*s)+(z*c)
            enddo
            z = sqrt(f*f + h*h)
            w(j) = z
            if (z.ne.0.0) then
               z = 1.0/z
               c = f*z
               s = h*z
            endif
            f = (c*g) + (s*y)
            x = -(s*g) + (c*y)
            do jj = 1, m
               y = a(jj,j)
               z = a(jj,i)
               a(jj,j) = (y*c) + (z*s)
               a(jj,i) = -(y*s) + (z*c)
            enddo
         enddo
         rv1(l) = 0.0
         rv1(k) = f
         w(k) = x
48       continue
3        continue
49    continue
      return
      end
C -------------------------------------------------------------
      subroutine chthres(nspts,irest,idir,singular,nd,emax)
       implicit double precision (a-h,o-z)
       character*5 plane(3)
       real tmp

       plane(1) = ' y-z '
       plane(2) = ' x-z '
       plane(3) = ' x-y '

       write(6,*) 'Do you want to change the threshold?(y/n)'
       write(6,103)
       if ((ans.eq.'y').or.(ans.eq.'Y')) then
       write(6,*)
       write(6,*) '-------------------'
       write(6,*) 'CHANGING THRESHOLDS'
       write(6,*) '-------------------'
       write(6,100) 'plane', 'threshold','values deleted', 'max eigen'
       write(6,100) '-----', '---------','--------------', '---------'
       write(6,101) plane(idir), singular, nd, nspts, emax
       write(6,*)
```

```
      irest = 1
 10       continue
      write(6,104) 'Enter threshold:'
      write(6,103)
      read(5,*) tmp
      write(6,*)
      write(6,*) 'You have entered:'
      write (6,105) 'Threshold:',tmp
      write(6,*) 'Is that correct(y/n)?'
      read(5,'(a)') ans
      if ((ans.eq.'y').or.(ans.eq.'Y')) then
        singular = tmp
         goto 99
          else
         goto 10
          endif
       endif
c
c          format statements
c
100    format (1x, a5, 3x, a9, 3x, a14, 3x, a9)
101    format (1x, a5, 3x, e8.1, 3x, '(',i3,'/',i3,')', 3x, f9.3)
103    format ('--->',$)
104    format(a,1x,i)
105    format(a,1x,i,e8.1)
c
99    continue
      ans='n'
      return
       end
c
c*********************************************************************
c
c                     SUBROUTINE MQ
c
c                Multiquadric-Biharmonic Method
c
c     Created:      SEP06,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine mq(naux,fo)
c
      implicit none
c
      integer          mqpt,NWK
      integer          imxs,jmxs,kmxs,nmxs
      integer          imxa,jmxa,kmxa,nmxa
c
      include 'csdcfd.par'
      include 'mq.par'
c
      integer          naux
```

```
      double precision fi(dnstr,3),fo(dnaro,3)
c
      double precision rmin,rmax
c
      integer          npi,npo
      double precision xi,yi,zi,ft
      double precision xo(dnaro),yo(dnaro),zo(dnaro),fa(dnaro)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     str common contains the structural surface data
c
c     nspts = total number of points on the surface
c     xi(dnstr) = Cartesian X location of data points
c     yi(dnstr) = Cartesian Y location of data points
c     zi(dnstr) = Cartesian Z location of data points
c     ft(dnstr) = Local data to be interfaced
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      common /str/ xi(dnstr),yi(dnstr),zi(dnstr),ft(dnstr+3),npi
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     aero common contains the aerodynamic (CFD) surface data
c
c     npts = total number of points on the surface
c     xo(dnaro) = Cartesian X location of data points
c     yo(dnaro) = Cartesian Y location of data points
c     zo(dnaro) = Cartesian Z location of data points
c     fa(dnaro) = Local data to be interfaced
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      common /aero/ xo,yo,zo,fa,npo
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     struct common contains the structural mode data
c
c     sxs(imxs,jmxs,kmxs) = Cartesian X location of data points
c     sys(imxs,jmxs,kmxs) = Cartesian Y location of data points
c     szs(imxs,jmxs,kmxs) = Cartesian Z location of data points
c     isg = total number of points along the x-direction
c     jsg = total number of points along the y-direction
c     ksg = total number of points along the z-direction
c     is  = points along the x-direction
c     js  = points along the y-direction
c     ks  = points along the z-direction
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      integer          is,js,ks,isg,jsg,ksg
      double precision sxs(imxs,jmxs,kmxs),sys(imxs,jmxs,kmxs)
      double precision szs(imxs,jmxs,kmxs)
c
      common /struct/sxs
     &,sys,szs
```

```
     &,is,js,ks,isg,jsg,ksg
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c.... Local variables
c
      integer           i,j,k,m
      integer           ni,no
      integer           ninterv
      integer           idiv,jdiv,kdiv
      integer           nscale
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c.... Dynamic Memory (real and integer in two separated spaces)
c
      integer           nr1,nr2,nr3,nr4,nr5,nr6,nr7,nr8
      integer           nr9,nr10,nr11,nr12,nr13,nr14,nr15
      integer           ni1,ni2,ni3,ni4
      integer           nexti,nextr,nmaxiv,nmaxrv
      integer           ivect(nnmaxi)
      double precision rvect(nnmaxr)
c
      common/point/     nexti,nextr,nmaxiv,nmaxrv
      common/memoi/     ivect
      common/memor/     rvect
c
      nmaxiv= nnmaxi
      nmaxrv= nnmaxr
c
      open(unit=27,file='rpar')
      read(27,*) rmin,rmax
      read(27,*) nscale
      close(27)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c.... Basic parameters calculation
c
c...... Total number of input (structural) points ---> (npi)
c
c...... Total number of output (aerodynamic) points ---> (npo)
c
c...... Maximum number of points within a sub-region
c
      ni = npi
      no = npo
c
c.... Estimate the number of subdivisions necessary in each direction
c
      idiv = isg/imax + 1
      jdiv = jsg/jmax + 1
      kdiv = ksg/kmax + 1
c
```

```
c.... and the total number of sub-regions
c
      ninterv = idiv*jdiv*kdiv
c
c.... Memory allocation
c
c...... Real part
c
c     nr1   ... xsi(ni,ninterv)
c     nr2   ... ysi(ni,ninterv)
c     nr3   ... zsi(ni,ninterv)
c     nr4   ... fsi(ni,3,ninterv)
c     nr5   ... xso(no,ninterv)
c     nr6   ... yso(no,ninterv,)
c     nr7   ... zso(no,ninterv,)
c     nr8   ... fso(no,3,ninterv)
c
c     nr9   ... xi2(npi)
c     nr10  ... yi2(npi)
c     nr11  ... zi2(npi)
c     nr12  ... xo2(npo)
c     nr13  ... yo2(npo)
c     nr14  ... zo2(npo)
c
      nr1   = 1
      nr2   = nr1 + ni*ninterv
      nr3   = nr2 + ni*ninterv
      nr4   = nr3 + ni*ninterv
      nr5   = nr4 + 3*ni*ninterv
      nr6   = nr5 + no*ninterv
      nr7   = nr6 + no*ninterv
      nr8   = nr7 + no*ninterv
      nextr = nr8 + 3*no*ninterv
c
      nr9   = nextr
      nr10  = nr9  + npi
      nr11  = nr10 + npi
      nr12  = nr11 + npi
      nr13  = nr12 + npo
      nr14  = nr13 + npo
      nr15  = nr14 + npo
c
c...... Integer part
c
c     ni1   ... conecti(npi)
c     ni2   ... conecto(npo,17)
c     ni3   ... npis(ninterv)
c     ni4   ... npos(ninterv)
c
      ni1   = 1
      ni2   = ni1 + npi
      ni3   = ni2 + 17*npo
      ni4   = ni3 + ninterv
      nexti = ni4 + ninterv
```

```
c
      write(*,*) 'Last allocated integer =',nexti
c
      write(*,*)'**********************************************'
      write(*,*)'PARAMETERS FOR THIS PROBLEM'
      write(*,*)
      write(*,*)'npi    npo        ',npi,npo
      write(*,*)'ni     no         ',ni,no
      write(*,*)'dnstr  dnaro      ',dnstr,dnaro
      write(*,*)'idiv   jdiv       ',idiv,jdiv
      write(*,*)'isg    jsg    ksg ',isg,jsg,ksg
      write(*,*)'is     js     ks  ',is ,js ,ks
      write(*,*)'imax   jmax   kmax',imax,jmax,kmax
      write(*,*)'idiv   jdiv   kdiv',idiv,jdiv,kdiv
      write(*,*)'nexti  nextr      ',nexti,nextr
      write(*,*)'nnmaxi nnmaxr     ',nnmaxi,nnmaxr
      write(*,*)'ninterv           ',ninterv
      write(*,*)
      write(*,*)'**********************************************'
      write(*,*)
c
      if(nmaxrv.lt.nr15.or.nmaxiv.lt.nexti) then
        write(*,*) 'NOT ENOUGH MEMORY FOR THIS CASE'
        write(*,*) 'Last allocated integer position    =',nexti
        write(*,*) 'Maximum allocated integer position =',nmaxiv
        write(*,*) 'Last allocated real position    =',nr15
        write(*,*) 'Maximum allocated real position =',nmaxrv
        if(nmaxrv.lt.nr15 ) STOP 'More real memory is necessary'
        if(nmaxiv.lt.nexti) STOP 'More integer memory is necessary'
      end if
c
c.... Put the input modes in a working format
c
      m=0
      do k=1,ksg
        do j=1,jsg
          do i=1,isg
            m=m+1
            fi(m,1) = sxs(i,j,k)
            fi(m,2) = sys(i,j,k)
            fi(m,3) = szs(i,j,k)
          enddo
        enddo
      enddo
c
c.... Transfer the working space and constants to the MQ solver
c
      call mq1 (dnaro,dnstr,npi,npo,ni,no,ninterv,idiv,jdiv,kdiv,
     1          rmin,rmax,naux,nscale,
     2          ivect(ni1),ivect(ni2),ivect(ni3),ivect(ni4),
     3          xi,yi,zi,fi,xo,yo,zo,fo,
     4          rvect(nr9), rvect(nr10),rvect(nr11),
     5          rvect(nr12),rvect(nr13),rvect(nr14),
     6          rvect(nr1),rvect(nr2),rvect(nr3),rvect(nr4),rvect(nr5),
```

```
     7            rvect(nr6),rvect(nr7),rvect(nr8) )
c
c.... END OF SUBROUTINE
c
      write(*,*)'END'
      write(*,*)
c
      return
      end
c
c***********************************************************************
c
c                        SUBROUTINE MQ1
c
c     Multiquadric-Biharmonic Method - Solution Routine
c
c     Created:        SEP07,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine mq1 (dnaro,dnstr,npi,npo,ni,no,ninterv,idiv,jdiv,kdiv,
     1                rmin,rmax,naux,nscale,
     2                conecti,conecto,npis,npos,
     3                xi0,yi0,zi0,fi,xo0,yo0,zo0,fo,
     4                xi,yi,zi,xo,yo,zo,
     5                xsi,ysi,zsi,fsi,xso,yso,zso,fso)
c
      implicit none
c
      integer          dnaro,dnstr
      integer          npi,npo,ni,no,ninterv,naux,nscale
      integer          idiv,jdiv,kdiv
      integer          conecti(npi),npis(ninterv)
      integer          conecto(npo,9),npos(ninterv)
      double precision rmin,rmax
      double precision xi0(dnstr),yi0(dnstr),zi0(dnstr)
      double precision xo0(dnaro),yo0(dnaro),zo0(dnaro)
      double precision xi(npi),yi(npi),zi(npi),fi(dnstr,3)
      double precision xo(npo),yo(npo),zo(npo),fo(dnaro,3)
      double precision xsi(ni,ninterv),ysi(ni,ninterv)
      double precision zsi(ni,ninterv),fsi(ni,3,ninterv)
      double precision xso(no,ninterv),yso(no,ninterv)
      double precision zso(no,ninterv),fso(no,3,ninterv)
c
c.... Local variables
c
      integer          i,interv
      integer          nr1,nr2,nr3,nr4,nr5,nr6,nr7,nr8,nr9
      integer          ni1
      integer          maxni,maxno,maxn
c
      integer          nexti,nextr,nnmaxi,nnmaxr
      integer          ivect(1)
```

```
      double precision rvect(1)
c
      common/point/    nexti,nextr,nnmaxi,nnmaxr
      common/memor/    rvect
      common/memoi/    ivect
c
c.... Put the input/output grids in a working format
c
      do i=1,npi
        xi(i) = xi0(i)
        yi(i) = yi0(i)
        zi(i) = zi0(i)
      end do
c
      do i=1,npo
        xo(i) = xo0(i)
        yo(i) = yo0(i)
        zo(i) = zo0(i)
      end do
c
c.... Scale the domain to a unity cube [0,1] x [0,1] x [0,1]
c
      if(nscale.eq.1) then
      write(*,*)'Scale the domain to a unity cube ...'
      write(*,*)
      end if
      call scalesr (nscale,npi,npo,naux,
     1              xi,yi,zi,xo,yo,zo)
c
c.... Partition the given domain
c
      write(*,*)'Partition the given domain (x-y-z plane) ...'
      write(*,*)
      call partit (idiv,jdiv,kdiv,npi,ni,npo,no,
     1             ninterv,dnstr,dnaro,
     2             xi,yi,zi,fi,xo,yo,zo,
     3             xsi,ysi,zsi,fsi,xso,yso,zso,
     4             conecti,conecto,npis,npos)
c
c.... Local memory allocation (real part)
c
      maxni = 0
      maxno = 0
      do i=1,ninterv
        write(*,*)'interv  npis(interv)',i,npis(i)
        write(*,*)'interv  npos(interv)',i,npos(i)
        if(npis(i).gt.maxni) maxni = npis(i)
        if(npos(i).gt.maxno) maxno = npos(i)
      end do
      if(maxni.gt.maxno) then
         maxn = maxni
      else
         maxn = maxno
      end if
```

```
c
c     nr1   ... B(maxni,maxni)
c     nr2   ... wkarear(maxn)
c     nr3   ... BIG(maxno,maxni)
c     nr5   ... R(maxni,naux)
c     nr7   ... RI(maxn,naux)
c     nr9   ... alpha(maxni)
c
      nr1   = nextr
      nr2   = nr1 + maxni*maxni
      nr3   = nr2 + maxn
      nr5   = nr3 + maxno*maxni
      nr7   = nr5 + maxni*naux
      nr9   = nr7 + maxn*naux
      nextr = nr9 + maxni
c
c.... Local memory allocation (integer part)
c
c     ni1   ... wkareai(maxn)
c
      ni1   = nexti
      nexti = ni1 + maxn
c
      if(nnmaxr.lt.nextr.or.nnmaxi.lt.nexti) then
        write(*,*) 'NOT ENOUGH MEMORY FOR THIS CASE'
        write(*,*) 'Last allocated integer position    =',nexti
        write(*,*) 'Maximum allocated integer position =',nnmaxi
        write(*,*) 'Last allocated real position    =',nextr
        write(*,*) 'Maximum allocated real position =',nnmaxr
        if(nnmaxr.lt.nextr) STOP 'More real memory is necessary'
        if(nnmaxi.lt.nexti) STOP 'More integer memory is necessary'
      end if
c
c.... Perform the MQ/TPS interpolation in each sub-region
c
      write(*,*)'Perform the interpolation in each sub-region ...'
      write(*,*)
      do interv = 1, ninterv
c
c...... Non-dimensionalization of the coordinates for a
c        given sub-region
c
        if(nscale.eq.1) then
        write(*,*)'Scaling sub-region', interv,' ...'
        write(*,*)
        end if
        call scalesr ( nscale,npis(interv),npos(interv),naux,
     1                 xsi(1,interv),ysi(1,interv),zsi(1,interv),
     2                 xso(1,interv),yso(1,interv),zso(1,interv) )
c
c...... Evaluate the interpolated function (fso) for the sub-region
c
        write(*,*)'Interpolating for sub-region', interv,' ...'
        write(*,*)
```

```
c
         call inter (maxni,maxno,maxn,naux,npis(interv),npos(interv),
     1              ni,no,rmin,rmax,
     2              xsi(1,interv),ysi(1,interv),zsi(1,interv),
     3              xso(1,interv),yso(1,interv),zso(1,interv),
     4              rvect(nr1),rvect(nr5),
     5              rvect(nr2),ivect(ni1),rvect(nr3),rvect(nr7),
     6              rvect(nr9),fsi(1,1,interv),
     7              fso(1,1,interv) )
c
      end do
c
c.... Average the interpolated function over the overlaped areas
c
      write(*,*)'Average over the overlaped areas ...'
      write(*,*)
      call overlap (dnaro,npo,no,ninterv,fso,conecto,
     1              fo)
c
c.... END OF SUBROUTINE
c
      return
      end
c
c
c**********************************************************************
c
c                        SUBROUTINE OVERLAP
c
c     Evaluate the value of the output function that belongs to an
c     overlaping area by simply weighting the interpolated result that
c     comes from the diferent sub-regions
c
c     Created:       SEP08,95
c     Last Modified:
c
c**********************************************************************
c
      subroutine overlap (dnaro,npo,nno,ninterv,fso,conecto,
     1                    fo)
c
      implicit         none
c
      integer          dnaro,npo,nno,ninterv
      integer          conecto(npo,17)
      double precision fso(nno,3,ninterv)
      double precision fo(dnaro,3)
c
      integer          m,i,nint,naux,interv,no
c
      do m=1,npo
        nint = conecto(m,1)
        if(nint.gt.17) STOP 'at overlap - nint > 9'
        do i=1,nint
```

```
            naux = 2*i
            interv = conecto(m,naux)
            no = conecto(m,naux+1)
            fo(m,1) = fo(m,1) + fso(no,1,interv)/nint
            fo(m,2) = fo(m,2) + fso(no,2,interv)/nint
            fo(m,3) = fo(m,3) + fso(no,3,interv)/nint
          end do
        end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c**********************************************************************
c
c                       SUBROUTINE SCALESR
c
c     Scales the sub-region to a unity cube [0,1] x [0,1] x [0,1]
c
c     Created:       SEP07,95
c     Last Modified:
c
c**********************************************************************
c
      subroutine scalesr (nscale,npi,npo,naux,
     1                    xsi,ysi,zsi,xso,yso,zso)
c
      implicit         none
c
      integer          nscale,npi,npo,naux
      double precision xsi(npi),ysi(npi),zsi(npi)
      double precision xso(npo),yso(npo),zso(npo)
c
      integer          i
      double precision xmin,xmax,ymin,ymax,zmin,zmax
      double precision auxx,auxy,auxz,dx,dy,dz
c
      double precision eps
      parameter (eps=1.0D-14)
c
c.... Scale the domain to a unit cube ( [0,1] x [0,1] x [0,1] )
c
c....... Find the min. and max. coordinate values along each direction
c        considering all the point involved in the sub-region (both input
c        as well as output points)
c
      xmax = xsi(1)
      xmin = xsi(1)
      ymax = ysi(1)
      ymin = ysi(1)
      zmax = zsi(1)
      zmin = zsi(1)
c
```

```
      call mimax (npi,xsi,ysi,zsi,npi,
     1            xmin,xmax,ymin,ymax,zmin,zmax)
c
      dx = xmax - xmin
      dy = ymax - ymin
      dz = zmax - zmin
c
c.... For the Thin-Plate Spline Method
c     (changes from 2-D to 3-D analysis)
c
c     naux = 2 --> 1-D sub-domain
c          = 3 --> 2-D sub-domain
c          = 4 --> 3-D sub-domain
c
        if(naux.eq.2.or.naux.eq.3.or.naux.eq.4) then
          if(dabs(dx).gt.eps) then
            if(dabs(dy).gt.eps) then
              if(dabs(dz).gt.eps) then
                 naux = 4
              else
                 naux = 3
              end if
            else
              naux = 2
            end if
          end if
        end if
c
c.... If scale is not to take place (nscale.ne.1), return to the
c     calling routine
c
      if(nscale.ne.1) RETURN
c
      call mimax (npo,xso,yso,zso,npo,
     1            xmin,xmax,ymin,ymax,zmin,zmax)
c
c...... All grid points have coordinates between [0,1]
c
c.... Note: If there is no variation along one direction,
c           it is made the constant equal to zero as part
c           of the scaling process
c
      dx = xmax - xmin
      dy = ymax - ymin
      dz = zmax - zmin
c
      if(DABS(dx).gt.eps) then
c
c.... Structural grid along x
c
        do i=1,npi
          xsi(i) = ( xsi(i) - xmin ) / dx
        end do
c
```

```
c.... Aerodynamic grid along x
c
        do i=1,npo
          xso(i) = ( xso(i) - xmin ) / dx
        end do
c
      else
c
c.... Structural grid along x
c
        do i=1,npi
          xsi(i) =  xsi(i) - xmin
        end do
c
c.... Aerodynamic grid along x
c
        do i=1,npo
          xso(i) =  xso(i) - xmin
        end do
c
      end if
c
c-----------------------------------------------------------------------
c
      if(DABS(dy).gt.eps) then
c
c.... Structural grid along y
c
        do i=1,npi
          ysi(i) = ( ysi(i) - ymin ) / dy
        end do
c
c.... Aerodynamic grid along y
c
        do i=1,npo
          yso(i) = ( yso(i) - ymin ) / dy
        end do
c
      else
c
c.... Structural grid along y
c
        do i=1,npi
          ysi(i) =  ysi(i) - ymin
        end do
c
c.... Aerodynamic grid along y
c
        do i=1,npo
          yso(i) =  yso(i) - ymin
        end do
c
      end if
c
```

```
c-----------------------------------------------------------------------
c
      if(DABS(dz).gt.eps) then
c
c.... Structural grid along z
c
        do i=1,npi
          zsi(i) = ( zsi(i) - zmin ) / dz
        end do
c
c.... Aerodynamic grid along z
c
        do i=1,npo
          zso(i) = ( zso(i) - zmin ) / dz
        end do
c
c.... For the Thin-Plate Spline Method
c     (changes from 2-D to 3-D analysis)
c
      else
c
c.... Structural grid along z
c
        do i=1,npi
          zsi(i) =  zsi(i) - zmin
        end do
c
c.... Aerodynamic grid along z
c
        do i=1,npo
          zso(i) =  zso(i) - zmin
        end do
c
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                        SUBROUTINE PARTIT
c
c     Performs the domain partition for the MQ method
c
c     Created:        SEP04,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine partit (idiv,jdiv,kdiv,npi,nni,npo,nno,
     1                   ninterv,dnstr,dnaro,
     2                   xi,yi,zi,fi,xo,yo,zo,
```

```
     3                    xsi,ysi,zsi,fsi,xso,yso,zso,
     4                    conecti,conecto,npis,npos)
c
      implicit none
c
      integer          idiv,jdiv,kdiv,npi,nni,npo,nno
      integer          ninterv,dnstr,dnaro
      double precision xi(npi),yi(npi),zi(npi),fi(dnstr,3)
      double precision xo(npo),yo(npo),zo(npo)
c
      integer          npis(ninterv),npos(ninterv)
      integer          conecti(npi),conecto(npo,17)
      double precision xsi(nni,ninterv),ysi(nni,ninterv)
      double precision zsi(nni,ninterv),fsi(nni,3,ninterv)
      double precision xso(nno,ninterv),yso(nno,ninterv)
      double precision zso(nno,ninterv)
c
      integer          ij,ji
c
      integer          i,j,k,m,interv,ni,no,ii,jj
      integer          naux,auxi,auxo
      double precision deltax,deltay,deltaz,tolx,toly,tolz
      double precision xmin,xmax,ymin,ymax,zmin,zmax
      double precision xmin0,ymin0,zmin0
c
c.... Initialize arrays
c
      do i=1,npi
        conecti(i) = 0
      end do
c
      do j=1,9
      do i=1,npo
        conecto(i,j) = 0
      end do
      end do
c
c.... Determine delta_x, delta_y, and delta_z based on the number of
c     subdivisions, as well as the size of the overlaping region based
c     on a percentage of the size of the interval (assumed to be 10%)
c
      xmax  = xi(1)
      xmin0 = xi(1)
      ymax  = yi(1)
      ymin0 = yi(1)
      zmax  = zi(1)
      zmin0 = zi(1)
c
      call mimax (npi,xi,yi,zi,npi,
     1             xmin0,xmax,ymin0,ymax,zmin0,zmax)
c
      call mimax (npo,xo,yo,zo,npo,
     1             xmin0,xmax,ymin0,ymax,zmin0,zmax)
c
```

```
      deltax = (xmax - xmin0)/dfloat(idiv)
      deltay = (ymax - ymin0)/dfloat(jdiv)
      deltaz = (zmax - zmin0)/dfloat(kdiv)
c
c..... This can only be used when one has a unit cube
c       deltax = 1.0/dfloat(idiv)
c       deltay = 1.0/dfloat(jdiv)
c       deltaz = 1.0/dfloat(kdiv)
c
      tolx = 0.1*deltax
      toly = 0.1*deltay
      tolz = 0.1*deltaz
c
c.... Find the association between a grid point and the given region
c
      interv = 0
c
      do k=1,kdiv
       zmin = zmin0 + (k-1)*deltaz - tolz
       zmax = zmin + deltaz + 2.0*tolz
c
       do j=1,jdiv
        ymin = ymin0 + (j-1)*deltay - toly
        ymax = ymin + deltay + 2.0*toly
c
        do i=1,idiv
        interv = interv + 1
        xmin = xmin0 + (i-1)*deltax - tolx
        xmax = xmin + deltax + 2.0*tolx
c
c...... Check which points are in the given region and assemble new
c       arrays for the coordinates
c
c... Check limits of the sub-region
c
        ni = 0
        do m=1,npi
          if(xi(m).ge.xmin.and.xi(m).le.xmax.and.
     1       yi(m).ge.ymin.and.yi(m).le.ymax.and.
     2       zi(m).ge.zmin.and.zi(m).le.zmax) then
            ni = ni + 1
            xsi(ni,interv) = xi(m)
            ysi(ni,interv) = yi(m)
            zsi(ni,interv) = zi(m)
            fsi(ni,1,interv) = fi(m,1)
            fsi(ni,2,interv) = fi(m,2)
            fsi(ni,3,interv) = fi(m,3)
c
c...... Save the connectivity information for the point
            naux = conecti(m)
            if(naux.gt.7) then
              write(*,*)'ERROR in point assignment to a sub-region'
              STOP 'Error-conecti'
            end if
```

```
              conecti(m) = naux + 1
c
            end if
          end do
          npis(interv) = ni
c
          no = 0
          do m=1,npo
            if(xo(m).ge.xmin.and.xo(m).le.xmax.and.
     1         yo(m).ge.ymin.and.yo(m).le.ymax.and.
     2         zo(m).ge.zmin.and.zo(m).le.zmax) then
              no = no + 1
              xso(no,interv) = xo(m)
              yso(no,interv) = yo(m)
              zso(no,interv) = zo(m)
c
c...... Save the connectivity information for the point
              naux = conecto(m,1)
              if(naux.gt.7) then
                write(*,*)'ERROR in point assignment to a sub-region'
                write(*,*)'point naux',m,naux
                write(*,*)'xo(m)  yo(m)',xo(m),yo(m)
                STOP 'Error-conecto'
              end if
              conecto(m,1) = naux + 1
              conecto(m,2*naux+2) = interv
              conecto(m,2*naux+3) = no
c
            end if
          end do
          npos(interv) = no
c
          end do
         end do
        end do
c
c.... Check if the original estimation of the number of intervals
c     (ninterv) has not been miscalculated
c
        if(interv.gt.ninterv) then
          write(*,*) 'From subroutine PARTIT'
          write(*,*) 'Bad estimation of total number of sub-regions'
          write(*,*) 'Estimated number (ninterv) =',ninterv
          write(*,*) 'Needed number (interv) =',interv
          STOP 'ERROR - ninterv too small'
        end if
c
c.... Check if all the points have been assigned to one of the regions
c
        auxi = 0
        do m=1,npi
          if(conecti(m).eq.0) then
            auxi = auxi + 1
           end if
```

```
      end do
c
      auxo = 0
      do m=1,npo
        if(conecto(m,1).eq.0) then
          auxo = auxo + 1
         end if
      end do
c
      if(auxi.ne.0.or.auxo.ne.0) then
        write(*,*)'ERROR in the domain sub-division'
        write(*,*)'Number of input points left over =', auxi
        write(*,*)'Number of output points left over =', auxo
        write(*,*)'Number of input points included =',npi - auxi
        write(*,*)'Number of output points included =',npo - auxo
        STOP 'domain sub-division'
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                       SUBROUTINE INTER
c
c     Performs the interpolation based on the MQ or TPS methods
c
c     naux = 1 --> Multiquadric Method
c          = 2 --> 1-D Thin-Plate Spline Method
c          = 3 --> 2-D Thin-Plate Spline Method
c          = 4 --> 3-D Thin-Plate Spline Method
c
c     Created:       SEP26,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine inter (ni,no,nmax,naux,npi,npo,nni,nno,rmin,rmax,
     1                   xi,yi,zi,xo,yo,zo,
     2                   B,R,wkarear,wkareai,BIG,RI,alpha,fi,
     3                   fo)
c
      implicit          none
c
      integer           ni,no,npi,npo,nni,nno
      integer           nmax,naux
      integer           wkareai(nmax)
      double precision rmin,rmax
      double precision xi(ni),yi(ni),zi(ni)
      double precision xo(no),yo(no),zo(no)
      double precision B(ni,ni),BIG(no,ni)
      double precision R(ni,naux),RI(nmax,naux)
```

```
      double precision wkarear(nmax),alpha(ni)
      double precision fi(nni,3)
      double precision fo(nno,3)
c
      integer          i,j
      double precision d
c
c.... Assembling the coefficient matrix [B]
c
c      write(*,*)'   Assembling the coefficient matrix [B]'
c      write(*,*)
c
      call bmatrx (naux,ni,npi,xi,yi,zi,ni,npi,xi,yi,zi,rmin,rmax,
     1             B)
c
c.... Assembling the coefficient matrix [R]
c
c      write(*,*)'   Assembling the coefficient matrix [R]'
c      write(*,*)
      call Rmatrx (ni,naux,ni,npi,xi,yi,zi,
     1             R)
c
c.... Perform first phase of the LU decomposition on matrix [B]
c
c      write(*,*)'   First phase of LU decomposition on [B]'
c      write(*,*)
      call ludcmpd (B,npi,ni,wkareai,d,wkarear)
c
c.... Do a loop over the modes (not implemented because the mode under
c                              consideration is defined at the driver
c                              level, which is not too efficient for
c                              this method)
c      do imode=1,mode
c
c.... Compute the interpolated mode for each of the components
c     of the given input mode
c
c...... x-component
c
      write(*,*)'   Compute the interpolated mode for x-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1            ni,npi,xi,yi,zi,rmin,rmax,
     2            B,R,wkarear,wkareai,BIG,RI,fi(1,1),alpha,
     3            fo(1,1))
c
c...... y-component
c
      write(*,*)'   Compute the interpolated mode for y-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1            ni,npi,xi,yi,zi,rmin,rmax,
     2            B,R,wkarear,wkareai,BIG,RI,fi(1,2),alpha,
     3            fo(1,2))
```

```
c
c...... z-component
c
      write(*,*)'   Compute the interpolated mode for z-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1            ni,npi,xi,yi,zi,rmin,rmax,
     2            B,R,wkarear,wkareai,BIG,RI,fi(1,3),alpha,
     3            fo(1,3))
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*********************************************************************
c
c                        SUBROUTINE FITFO
c
c     Performs the interpolation based on the MQ or TPS methods
c
c     Created:        SEP26,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine fitfo (nmax,naux,no,npo,xo,yo,zo,
     1                  ni,npi,xi,yi,zi,rmin,rmax,
     2                  B,R,wkarear,wkareai,BIG,RI,fi,alpha,
     3                  fo)
c
      implicit         none
c
      integer          nmax,naux
      integer          ni,npi,no,npo
      integer          wkareai(nmax)
      double precision rmin,rmax
      double precision xi(ni),yi(ni),zi(ni)
      double precision xo(no),yo(no),zo(no)
      double precision B(ni,ni),BIG(no,ni)
      double precision R(ni,naux),RI(nmax,naux)
      double precision alpha(ni),wkarear(nmax)
      double precision fi(ni)
      double precision fo(no)
c
      integer          i,j
      integer          aux2(16)
      double precision err,err2,d
      double precision beta(4)
      double precision auxf(4),aux1(4,4),aux3(16)
c
c.... Backup {fi} in wkarear
c
      do i=1,npi
```

```
          wkarear(i) = fi(i)
        end do
c
c.... Perform second phase of LU decomposition (backsubstitution)
c
c        write(*,*)'   ... Perform second phase of LU decomposition [B]'
c        write(*,*)
        call lubksbd (B,npi,ni,wkareai,fi)
c
c.... Calculate the constants \alpha and
c                          constraint constants \beta
c
c       beta = ( R^T . (B^{-1}.R) )^{-1} . R^T . fi
c
c.... Backup [R] in [R_I]
c
        do j=1,naux
        do i=1,npi
          RI(i,j) = R(i,j)
        end do
        end do
c
c...... Perform second phase of LU decomposition (backsubstitution)
c
c        write(*,*)'   ... Perform second phase of LU decomposition [R]'
c        write(*,*)
        do i=1,naux
          call lubksbd (B,npi,ni,wkareai,RI(1,i))
        end do
c
        call lsmptd (R,ni,npi,naux,RI,nmax,naux,aux1,4)
c
        call lsmptd (R,ni,npi,naux,fi,ni,1,auxf,4)
c
        if(naux.eq.1) then
c
c...... For Multiquadric Method
          beta(1) = auxf(1) / aux1(1,1)
c
          do i=1,npi
            alpha(i) = fi(i) - RI(i,1)*beta(1)
          end do
c
        elseif(naux.eq.2.or.naux.eq.3.or.naux.eq.4) then
c
c...... For Thin-Plate Spline Method
c          write(*,*)'   ... Perform first phase decomposition for [aux1]'
c          write(*,*)
          call ludcmpd (aux1,naux,4,aux2,d,aux3)
c          write(*,*)'   ... Perform second phase decomposition for [aux1]'
c          write(*,*)
          call lubksbd (aux1,naux,4,aux2,auxf)
c          call lubksbd (aux1,3,4,aux2,auxf)
          do i=1,naux
```

```
          beta(i) = auxf(i)
        end do
        call lsmpd (RI,nmax,npi,naux,beta,4,1,alpha,ni)
c
        do i=1,npi
          alpha(i) = fi(i) - alpha(i)
        end do
c
      end if
c
c.... Check on the constraints
c     (the constraint requires that R^T . alpha = 0)
c
      call lsmptd (R,ni,npi,naux,alpha,ni,1,auxf,4)
      write(*,*)'    ... Error in the constraint for alpha'
      do i=1,naux
        write(*,*)'         constraint',i,' = ',auxf(i)
      end do
      write(*,*)
c
c.... Evaluate the coefficient matrix based on both input and output
c     grid points
c
c      write(*,*)'    ... Evaluate [B_IG]'
      call bmatrx (naux,no,npo,xo,yo,zo,ni,npi,xi,yi,zi,rmin,rmax,
     1             BIG)
c
c      write(*,*)'    ... Evaluate [R_I]'
c      write(*,*)
      call Rmatrx (nmax,naux,no,npo,xo,yo,zo,
     1             RI)
c
c.... Evaluate the product [BIG]*{alpha} + [RI]*{beta} = {Hi}
c
c      write(*,*)'    ... Evaluate {Hi}'
c      write(*,*)
      call lsmpd (BIG,no,npo,npi,alpha,ni,1,fo,no)
      call lsmpd (RI,nmax,npo,naux,beta,naux,1,wkarear,nmax)
      do i=1,npo
         fo(i) = fo(i) + wkarear(i)
      end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*********************************************************************
c
c                        SUBROUTINE BMATRX
c
c     Generates the coefficient matrix based on the basis functions and
c     a user-defined varing parameter "r"
c
```

```
c     Created:       SEP04,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine bmatrx (naux,no,npo,xo,yo,zo,ni,npi,xi,yi,zi,rmin,rmax,
     1                   B)
c
      implicit         none
c
      integer          naux,ni,no,npi,npo
      double precision rmin,rmax
      double precision xo(no),yo(no),zo(no)
      double precision xi(ni),yi(ni),zi(ni)
      double precision B(no,ni)
c
      integer          i,j
      double precision npi1,exp,r2,raux
      double precision xxi,xxj,yyi,yyj,zzi,zzj
c
      double precision eps
      parameter (eps=1.0d-14)
c
c.... Check if MQ or TPS has been selected
c
c---------------------------------------------------------------------
c     Multiquadrics Basis Functions
c---------------------------------------------------------------------
      if(naux.eq.1) then
c
c.... Constant calculation for the varying "r" parameter
c
      npi1 = dfloat(npi) - 1.0
      raux = (rmax/rmin)*(rmax/rmin)
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        yyj = yi(j)
        zzj = zi(j)
        exp = ( dfloat(j) - 1.0 ) / npi1
        r2 = (rmin*rmin)*( (raux)**exp )
        do i=1,npo
          xxi = xo(i)
          yyi = yo(i)
          zzi = zo(i)
          B(i,j) = dsqrt( (xxi - xxj)*(xxi - xxj) +
     1                    (yyi - yyj)*(yyi - yyj) +
     2                    (zzi - zzj)*(zzi - zzj) + r2 )
        end do
      end do
c
      return
```

```
      end if
c
c-------------------------------------------------------------------
c     End of MQ
c-------------------------------------------------------------------
c
c-------------------------------------------------------------------
c     Thin-Plate Spline Basis Functions
c-------------------------------------------------------------------
c
c.... 1-D Problems
c
      if(naux.eq.2) then
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        zzj = zi(j)
        do i=1,npo
          xxi = xo(i)
          zzi = zo(i)
          r2 =  (xxi - xxj)*(xxi - xxj) +
     1          (zzi - zzj)*(zzi - zzj)
          if(r2.lt.eps) then
            B(i,j) = 0.0
          else
            B(i,j) = r2 * DLOG(r2) / 2.0
          end if
        end do
      end do
c
      return
      end if
c
c.... 2-D Problems
c
      if(naux.eq.3) then
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        yyj = yi(j)
        do i=1,npo
          xxi = xo(i)
          yyi = yo(i)
          r2 =  (xxi - xxj)*(xxi - xxj) +
     1          (yyi - yyj)*(yyi - yyj)
          if(r2.lt.eps) then
            B(i,j) = 0.0
          else
            B(i,j) = r2 * DLOG(r2) / 2.0
          end if
```

```
          end do
        end do
c
        return
        end if
c
c.... 3-D Problems
c
        if(naux.eq.4) then
c
c.... Evaluate the matrix of coefficients [B]
c
        do j=1,npi
          xxj = xi(j)
          yyj = yi(j)
          zzj = zi(j)
          do i=1,npo
            xxi = xo(i)
            yyi = yo(i)
            zzi = zo(i)
            r2 =  (xxi - xxj)*(xxi - xxj) +
     1            (yyi - yyj)*(yyi - yyj) +
     2            (zzi - zzj)*(zzi - zzj)
            if(r2.lt.eps) then
              B(i,j) = 0.0
            else
              B(i,j) = r2 * DLOG(r2) / 2.0
            end if
          end do
        end do
c
        return
        end if
c------------------------------------------------------------------------
c     End of TPS
c------------------------------------------------------------------------
c
        if(naux.ne.1.and.naux.ne.2.and.naux.ne.3.and.naux.ne.4) then
          write(*,*)'naux not properly defined in subroutine "bmatrx"'
          write(*,*)'naux =',naux
          STOP 'naux not equal to 1, 2, 3 or 4 in bmatrx'
        end if
c
c.... END OF SUBROUTINE
c
        return
        end
c
c************************************************************************
c
c                          SUBROUTINE RMATRX
c
c
c     Generates the constraint matrix based on the basis functions
c
```

```
c     Created:        SEP26,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine Rmatrx (no,naux,ni,npi,xi,yi,zi,
     1                   R)
c
      implicit          none
c
      integer           ni,no,npi,naux
      double precision xi(ni),yi(ni),zi(ni)
      double precision R(no,naux)
c
      integer           i
c
c.... Evaluate the matrix of constraints [R]
c
      do i=1,npi
        R(i,1) = 1.0
      end do
c
      if(naux.eq.2) then
        do i=1,npi
          R(i,2) = xi(i)
        end do
      end if
c
      if(naux.eq.3) then
        do i=1,npi
          R(i,2) = xi(i)
          R(i,3) = yi(i)
        end do
      end if
c
      if(naux.eq.4) then
        do i=1,npi
          R(i,2) = xi(i)
          R(i,3) = yi(i)
          R(i,4) = zi(i)
        end do
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                         SUBROUTINE FOTSXA
c
c     Transfer output data from the simple array used in MQ calculations
c     to the grid format of the driver (sxs,sys,szs)
```

```
c
c     Created:        SEP04,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine fotsxa (npo,fo,ia,ja,ka,mode,sca)
c
      implicit     none
      integer      ia,ja,ka,mode
      integer      npo
      real         fo(npo)
      real         sca(ia,ja,ka,mode)
c
      integer      i,j,k,m
c
      m=1
      do k=1,ka
       do j=1,ja
        do i=1,ia
          sca(i,j,k,mode)=fo(m)
          m=m+1
        enddo
       enddo
      enddo
c
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*********************************************************************
c
c                       SUBROUTINE MIMAX
c
c     Finds the minimum and the maximum among the elements of a vector
c
c     Created:        OCT24,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine mimax (npi,xsi,ysi,zsi,ni,
     1                  xmin,xmax,ymin,ymax,zmin,zmax)
c
      implicit         none
      integer          ni,npi
      double precision xsi(ni),ysi(ni),zsi(ni)
      double precision xmin,xmax,ymin,ymax,zmin,zmax
c
      integer          i
      double precision auxx,auxy,auxz
c
```

```
c....... Find the min. and max. coordinate values along each direction
c         considering all the point involved in the sub-region (both input
c         as well as output points)
c
      do i=1,npi
        auxx = xsi(i)
        auxy = ysi(i)
        auxz = zsi(i)
        if(auxx.gt.xmax) xmax = auxx
        if(auxx.lt.xmin) xmin = auxx
        if(auxy.gt.ymax) ymax = auxy
        if(auxy.lt.ymin) ymin = auxy
        if(auxz.gt.zmax) zmax = auxz
        if(auxz.lt.zmin) zmin = auxz
      end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c---------------------------------------------
c This subroutine performs Lower - Upper (LU) decomposition
c
c Obtained From:
c   "Numerical Recipes", By  Press & Flannery et al.,
c      Cambridge University Press, 1989, p. 35-36
c a = matrix logical dimensions m by n
c     physical dimensions mp by np
c xx - row permutation effected by partial pivoting
c vv - internal
c------------------------------------------------------------------
      SUBROUTINE LUDCMPd(A,N,NP,INDX,D,VV)
      implicit double precision (a-h,o-z)
      PARAMETER (NMAX=100,TINY=1.0E-20)
c      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      DIMENSION A(NP,NP),INDX(N),VV(NP)
      D=1.
      DO 12 I=1,N
        AAMAX=0.
        DO 11 J=1,N
          IF (dABS(A(I,J)).GT.AAMAX) AAMAX=dABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        DO 14 I=1,J-1
          SUM=A(I,J)
          DO 13 K=1,I-1
            SUM=SUM-A(I,K)*A(K,J)
13        CONTINUE
          A(I,J)=SUM
14      CONTINUE
```

```
          AAMAX=0.
          DO 16 I=J,N
            SUM=A(I,J)
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
15          CONTINUE
            A(I,J)=SUM
            DUM=VV(I)*dABS(SUM)
            IF (DUM.GE.AAMAX) THEN
              IMAX=I
              AAMAX=DUM
            ENDIF
16        CONTINUE
          IF (J.NE.IMAX)THEN
            DO 17 K=1,N
              DUM=A(IMAX,K)
              A(IMAX,K)=A(J,K)
              A(J,K)=DUM
17          CONTINUE
            D=-D
            VV(IMAX)=VV(J)
          ENDIF
          INDX(J)=IMAX
          IF(A(J,J).EQ.0.)A(J,J)=TINY
          IF(J.NE.N)THEN
            DUM=1./A(J,J)
            DO 18 I=J+1,N
              A(I,J)=A(I,J)*DUM
18          CONTINUE
          ENDIF
19      CONTINUE
        RETURN
        END
c
c-------------------------------------
c This subroutine performs Lower - Upper (LU) backsubstitution
c
c Obtained From:
c   "Numerical Recipes", By  Press & Flannery et al.,
c      Cambridge University Press, 1989, p. 37
c a = matrix logical dimensions m by n
c     physical dimensions mp by np
c xx - row permutation effected by partial pivoting
c b - on input contains contain RHS of A x = b
c     on output contains the solution x
c---------------------------------------------------------------
        SUBROUTINE LUBKSBd(A,N,NP,INDX,B)
        implicit double precision (a-h,o-z)
        DIMENSION A(NP,NP),INDX(N),B(N)
        II=0
        DO 12 I=1,N
          LL=INDX(I)
          SUM=B(LL)
          B(LL)=B(I)
```

```
        IF (II.NE.0)THEN
          DO 11 J=II,I-1
            SUM=SUM-A(I,J)*B(J)
11        CONTINUE
        ELSE IF (SUM.NE.0.) THEN
          II=I
        ENDIF
        B(I)=SUM
12    CONTINUE
      DO 14 I=N,1,-1
        SUM=B(I)
        DO 13 J=I+1,N
          SUM=SUM-A(I,J)*B(J)
13      CONTINUE
        B(I)=SUM/A(I,I)
14    CONTINUE
      RETURN
      END

C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                       *
C*                    MULTIPLICATION OF TWO MATRICES                     *
C*                                                                       *
C*           Last Update: Nov 04, 92                                     *
C*                                                                       *
C*  OBJECTIVE :                                                          *
C*    Multiplication of A.B = C                                          *
C*                                                                       *
C*  INPUT :                                                              *
C*    A       : matrix n vs. m                                           *
C*    B       : matrix m vs. l                                           *
C*                                                                       *
C*  OUTPUT :                                                             *
C*    C       : matrix n vs. l, result of A.B                            *
C*                                                                       *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      subroutine lsmpd (a,na,n,m,b,nb,l,c,nc)
c
      implicit         none
      integer          n,m,l,na,nb,nc
      double precision a(na,m),b(nb,l)
      double precision c(nc,l)
c
      integer          i,j,k
      double precision aux
c
      do 200 i = 1,n
      do 200 j = 1,l
     aux = 0.0d0
     do 100 k = 1,m
        aux = aux + a(i,k) * b(k,j)
 100      continue
     c(i,j) = aux
```

```
 200  continue
c
      return
      end
C
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                        *
C*                  MULTIPLICATION OF TWO MATRICES                        *
C*                     [A] in band storage mode                           *
C*                                                                        *
C*                                                                        *
C*            Last Update: Dec 16, 92                                     *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                        *
C*  OBJECTIVE :                                                           *
C*    Multiplication of A.B = C                                           *
C*                                                                        *
C*  INPUT :                                                               *
C*    A       : matrix n vs. n (BAND STORAGE MODE)                        *
C*    B       : matrix n vs. l                                            *
C*                                                                        *
C*  OUTPUT :                                                              *
C*    C       : matrix n vs. l, result of A.B                             *
C*                                                                        *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      subroutine lsmpbd (a,n,band,b,l,c)
c
      implicit          none
c
      integer           n,band,l
      double precision  a(n,band), b(n,l)
      double precision  c(n,l)
c
      integer           i,j,k
      double precision  aux,baux
c
      do 100 j=1,l
      do 100 i=1,n
         c(i,j) = 0.0d0
 100  continue
c
      do 300 k=1,l
      do 300 i=1,n
     aux = 0.0d0
         do 200 j=1,band
        aux = aux + b(i+j-1,k)*a(i,j)
 200     continue
     c(i,k) = aux
 300  continue
c
      do 400 k=1,l
        do 400 i=1,n-band+1
          baux = b(i,k)
```

```
        do 400 j=2,band
       c(i-1+j,k) = c(i-1+j,k) + baux*a(i,j)
 400  continue
c
      do 500 k=1,l
      do 500 i=n-band+2,n-1
      baux = b(i,k)
      do 500 j=2,n-i+1
       c(i-1+j,k) = c(i-1+j,k) + baux*a(i,j)
 500  continue
c
      return
      end
c
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                    *
C*                MULTIPLICATION OF TWO MATRICES                      *
C*                                                                    *
C*          Last Update: Mar 09, 93                                   *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                    *
C*  OBJECTIVE :                                                       *
C*    Multiplication of A^T.B = C                                     *
C*                                                                    *
C*  INPUT :                                                           *
C*    A       : matrix n vs. m                                        *
C*    B       : matrix n vs. l                                        *
C*                                                                    *
C*  OUTPUT :                                                          *
C*    C       : matrix m vs. l, result of A.B                         *
C*                                                                    *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      subroutine lsmptd (a,na,n,m,b,nb,l,c,nc)
c
      implicit         none
      integer          n,m,l,na,nb,nc
      double precision a(na,m),b(nb,l)
      double precision c(nc,l)
c
      integer          i,j,k
      double precision aux
c
      do 200 i = 1,m
      do 200 j = 1,l
     aux = 0.0d0
     do 100 k = 1,n
        aux = aux + a(k,i) * b(k,j)
 100     continue
     c(i,j) = aux
 200  continue
c
      return
      end
```

```
      subroutine nurbs(mode,f)

      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(ndom=2,ndim=6,ndep=6,nwork=NWK)
      parameter(maxc=(ndim+1)*imxs*jmxs)
c
c       xs,ys,zs     - structural grid points
c       sxs,sys,szs - structural mode shapes values at grid points
c       xa,ya,za     - aerodynamic grid points
c       sxa,sya,sza - calculated aerodynamic deflections
c       isg,jsg,ksg - number of structural points
c       is,js,ks     - number of structural points ( for each mode)
c       ia,ja,ka     - number of aerodynamic points
c
c
        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
       common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       common /cnt/ ia,ja,ka
      common /para/ npt_s,npt_t,sdir(imxs*jmxs),tdir(jmxs)
c
c Local Variables
c     uo - array for aerodynamic grid points and mode shapes
c     u  - array for structural grid points and mode shapes
c     f  - array for structural grid points to be parameterized
c     st - parameterized values for the structural grid
c     sttest - parameterized value of the aerodynamic grid
c     work   - scratch array for dtnurbs subroutines
c     NWK    - dimension of scratch array
c     v      - value of spline at wanted location
c     out1,2,3 -  holding arrays for spline output values
c     ipdir - holding array for the plane direction
c
c
      double precision uo(imxa ,jmxa ,ndim), v(10)
      double precision work(NWK), u(imxs,jmxs,ndim),snpvl(2),
     &pt(3),p(2),s_carray(maxc)
       dimension f(dnaro,3)
c         for fitsurf
      dimension c_tmp((ndim+1)*jmxs*imxs)
      integer npts_curv(jmxs)
c
      integer ipdir(dnaro)
      dimension xd(4),yd(4),zd(4),fd(4),tmp(dnstr),tmp2(dnstr)

c----------------------------------------------------------------------
c Logicals
c----------------------------------------------------------------------
c
c  npi   - number of point in CSD grid
c  ndim  - number of dependant variables
```

```
c  ndeg  - degree of spline
c
      ipar=2

c       write(*,*) ' Enter the spline degree '
c       read(*,*) ndeg
c      write(*,*)' Enter the spline degree in the s direction'
c      read(*,*)ndeg(1)
c      write(*,*)' Enter the spline degree in the t direction'
c      read(*,*)ndeg(2)
c      ndeg(1)=3
c      ndeg(2)=3
       ndeg=3
c
c  Rearrange structural grid and mode information
c
      m=0
      do k=1,ksg
        do j=1,jsg
          do i=1,isg
            m=m+1
            u(i,j,1)=xt(m)
            u(i,j,2)=yt(m)
            u(i,j,3)=zt(m)
            u(i,j,4)=sxs(i,j,k)
            u(i,j,5)=sys(i,j,k)
c            u(i,j,4)=0.0
c            u(i,j,5)=0.0
            u(i,j,6)=szs(i,j,k)
          enddo
        enddo
      enddo
c
c Rearrange aerodynamic grid information
c
      m=0
      do k=1,ka
        do j=1,ja
          do i=1,ia
            m=m+1
            uo(i,j,1)=xa(m)
            uo(i,j,2)=ya(m)
            uo(i,j,3)=za(m)
          enddo
        enddo
      enddo
c
c     Use surface fitting routine for non-regular CSD grid
c
c =========================================================================
c
c...this program will fit a NURB surface to offset points.
c
c  Version:     6.0   Using DT_NURBS for fitting
```

```
c  Date:         2/1991
c  Programmer:  Bob Ames, DTRC
c  Modified for use in FASIT 8/95 & 9/95 by MJ Smith
c
c  Process...
c  Offset data is read in for each curve.  Each curve may have any number
c
c MEMORY:
c This program can COMSUME memory if you're not careful!  See fitsurf.inc.
c
c
c =========================================================================


c ***
c Start loop for each curve
c ***
         do  j=1,jsg

c Get number of points...check limits
c
           npts_curv(j)=isg
c
           if (npts_curv(j) .gt. imxs) then
               write(6,*) ' Error max mumber of points per '
               write(6,*) ' curve exceeded!  Terminating..!'
               stop
           end if

c Find curve with maximum number of points
           if(j .eq. 1) then
              maxpts= npts_curv(j)
           else
              maxpts=max0(npts_curv(j), maxpts)
           end if
c
           do  k=1,npts_curv(j)
c Check for coincident points
             if (k .gt. 1) then
c Assume points coincident unless proven wrong
                icoin=0
                do i=1,ndep
                   if (u(k,j,i).eq.u(k-1,j,i)) then
                       icoin=icoin+1
                   else
                       continue
                   end if
                end do
                if (icoin.eq.ndep) then
                  write(6,*) 'Error... input file'
                  write(6,*) 'contains coincident points'
                  write(6,*) ' '
c                 write(6,fmt='('Curve=',i3,' Point=',i3)')
c    &                                  j,k
```

```
                stop
              end if

           end if
         end do       !end npts
       end do         !end ncurv
c
c            Fit surface to input data
c
      call surface(jsg,npts_curv,ndeg,maxpts,u,s_carray)
       do i=1,100
         write(6,*) ' s_carray ',i,s_carray(i)
       enddo
c
c            end of original fitsurf program
c
c*******************************************************************
c            Manipulate the Unknown Function Grid to Determine
c                 the Parameterization
c*******************************************************************
        m=0
        do i=1,ia-1
        do j=1,ja-1
           m=m+1
           jp1=i*ia+j
           xd(1)=xa(m)
           yd(1)=ya(m)
           zd(1)=za(m)
           xd(2)=xa(m+1)
           yd(2)=ya(m+1)
           zd(2)=za(m+1)
           xd(3)=xa(jp1)
           yd(3)=ya(jp1)
           zd(3)=za(jp1)
           xd(4)=xa(jp1+1)
           yd(4)=ya(jp1+1)
           zd(4)=za(jp1+1)
           call plane1(xd,yd,zd,4,ipdir(m),iplane)
           write(6,*) ' plane', m,ipdir(m),iplane
        enddo
        enddo
        do m=1,npts
c
c            Now search for the known panel which encloses
c            the point, based on the direction of the data panel
c
        call search(ipdir(m),lout,mout,iout,jout,
     1       xa(m),ya(m),za(m),xd,yd,zd)
        write(6,*) ' search ',m,lout,mout,iout,jout
c
c            Now call the bivariate interpolation or extrapolation
c                routines
c
c                   interpolation
```

```
      incx=1
      if(lout.gt.0) then
        fd(1)=sdir(mout)
        fd(2)=sdir(mout+1)
        fd(4)=sdir(mout+isg)
        fd(3)=sdir(mout+isg+1)
        xx=xa(m)
        yy=ya(m)
        do ll=1,4
          write(6,*) ' 1 fd ',xd(ll),yd(ll),fd(ll)
        enddo
        call bivarin(xd,yd,fd,xx,yy,z,ierr)
        write(6,*) ' xx ',xx,yy,z
        snpvl(1)=z
        fd(1)=tdir(jout)
        fd(2)=tdir(jout)
        fd(4)=tdir(jout+1)
        fd(3)=tdir(jout+1)
        xx=xa(m)
        yy=ya(m)
        call bivarin(xd,yd,fd,xx,yy,z,ierr)
        snpvl(2)=z
        write(6,*) ' calling dtnpvl ',snpvl(1),snpvl(2)
        call dtnpvl(snpvl,incx,s_carray,work,NWK,v,ier)
        do i=1,4
         write(6,*) ' int ',xd(i),yd(i),fd(i)
        enddo
        do i=1,6
          write(6,*) ' dtnpvl ',i,v(i)
        enddo
        f(m,1)=v(4)
        f(m,2)=v(5)
        f(m,3)=v(6)
       else if (lout.eq.-1) then
c               extrapolate along x direction
c        do l=1,3
          l=3
          mout=((jout-1)*isg)+iout
          mp1=(jout*isg)+iout
          do ll=1,isg
          tmp(ll)=szs(ll,jout,1)
          tmp2(ll)=xt((jout-1)*isg+ll)
          write(101,*) ' a ',ll,tmp(ll),tmp2(ll)
          enddo
          xx=xa(m)
          call polint(tmp2,tmp,isg,xx,fd(1),err)
          if(l.eq.3) then
          write(101,*) 'polint ',m, xa(m),fd(1)
          write(101,*) (xt(ll),ll=nout,nout+isg),
     1       (tmp(i),i=1,isg)
          endif
c
          do ll=1,isg
          tmp(ll)=szs(ll,jout+1,1)
```

```
          tmp2(ll)=xt(jout*isg+ll)
          write(101,*) ' b ',ll,tmp(ll),tmp2(ll)
          enddo
          xx=xa(m)
          call polint(tmp2,tmp,isg,xx,fd(4),err)
          xd(1)=xa(m)
          xd(2)=xt(mout)
          xd(3)=xt(mp1)
          xd(4)=xa(m)
          yd(1)=yt(mout)
          yd(2)=yt(mout)
          yd(3)=yt(mp1)
          yd(4)=yt(mp1)
          fd(2)=u(iout,jout,l+3)
          fd(3)=u(iout,jout+1,l+3)
          xx=xa(m)
          yy=ya(m)
          call bivarin(xd,yd,fd,xx,yy,z,ierr)
          f(m,l)=z
c         enddo
        else if (lout.eq.-2) then
c             extrapolate along y direction
c        do l=1,3
          l=3
          mout=((jout-1)*isg)+iout
          do ll=1,jsg
          tmp(ll)=szs(iout,ll,1)
          tmp2(ll)=yt((ll-1)*isg+iout)
          enddo
          yy=ya(m)
          call polint(tmp2,tmp,jsg,yy,fd(1),err)
          if(l.eq.3) then
c          write(6,*) 'polint ',m, xa(m),fd(1)
c          write(6,*) (xt(ll),ll=nout,nout+isg),
c    1       (tmp(i),i=1,isg)
          endif
c
          do ll=1,jsg
          tmp(ll)=szs(iout+1,ll,1)
          tmp2(ll)=yt((ll-1)*isg+iout+1)
          enddo
          yy=ya(m)
          call polint(tmp2,tmp,jsg,yy,fd(4),err)
          mout=((jout-1)*isg+iout)
          xd(1)=xt(mout)
          xd(2)=xt(mout)
          xd(3)=xt(mout+1)
          xd(4)=xt(mout+1)
          yd(1)=ya(m)
          yd(2)=yt(mout)
          yd(3)=yt(mout+1)
          yd(4)=ya(m)
          fd(2)=u(iout,jout,l+3)
          fd(3)=u(iout+1,jout,l+3)
```

```
               xx=xa(m)
               yy=ya(m)
               call bivarin(xd,yd,fd,xx,yy,z,ierr)
               f(m,l)=z
c              enddo
            else if (lout.eq.-3) then
c                   extrapolate in a corner
c             do l=1,3
               l=3
               mout=((jout-1)*isg)+iout
               mp1=(jout*isg)+iout
               do ll=1,isg
               tmp(ll)=szs(ll,jout,l)
               tmp2(ll)=xt((jout-1)*isg+ll)
               enddo
               xx=xa(m)
               call polint(tmp2,tmp,isg,xx,fd(1),err)
               if(l.eq.3) then
c               write(6,*) 'polint ',m, xa(m),fd(1)
c               write(6,*) (xt(ll),ll=nout,nout+isg),
c     1            (tmp(i),i=1,isg)
               endif
c
               do ll=1,jsg
               tmp(ll)=szs(iout,ll,l)
               tmp2(ll)=yt((ll-1)*isg+iout)
               enddo
               yy=ya(m)
               call polint(tmp2,tmp,jsg,yy,fd(4),err)
               f(m,l)=((fd(1)-szs(iout,jout,l))+
     1           (fd(4)-szs(iout,jout,l)))+szs(iout,jout,l)
               write(102,*) ' fd ',m,l,f(m,l)
c               end do m=1,npts
           endif
           enddo
c
           do i=1,dnaro
            write(105,*) i,f(i,3)
           enddo
            return
            end


      subroutine surface (ncurv,npts_curv,ndeg,maxpts,xyz,s_carray)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
c
c  Date:        7/1990
c  Programmer:  Bob Ames, DTRC
c
c
c  --------------------
c  Variable definitions
c  --------------------
```

```
c
c  carray    - array holding nurb data
c  c_tmp     - temp holding array of individual curves
c  hold      - scratch space for DT_NURBS
c  maxpts    - maximum number of points for any input curve(i)
c  ndeg      - degree of curve or surface (order=ndeg+1)
c  ndep      - number of dependant variables allowed (x,y,z,cp,vx,vy,vz...)
c  ndom      - number of independant variables (s,t...)
c  npt_gpar  - number of data points for each ndom
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C
c Include parameters
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(ndom=2,ndim=6,ndep=6,nwork=NWK)
c DTGPAR variables
      common /para/ npts,nptt,sdir(imxs*jmxs),tdir(jmxs)
      parameter(maxc=(ndim+1)*imxs*jmxs)
      dimension xyz(imxs,jmxs,ndim)
      dimension temp(imxs*ndep),c_tmp(maxc),s_carray(maxc)
      dimension t_gpar(imxs*ndom*ndep),s_gpar(jmxs*ndom*ndep)
      dimension c_carray(maxc),hold(NWK),work(NWK)
      integer npt_gpar(ndom),ipntr_s(jmxs*ndep*2),len_c
      integer  npts_curv(jmxs),icc

      logical diag

c ***
c Set initial values
c ***

c Other logicals

       diag= .false.
       diag= .true.
c ***********************************************************************
c Generate parametric array of initial input data in preparation for
c curve fitting
c ***********************************************************************
c ***
c Load temp array with input data from data base
c ***
      ipntr_s(1)=1
      ndep_gpar=3
      npts=0
      nptt=ncurv
      do j=1,ncurv
         index=1
         do m=1,ndep_gpar
            do k=1,npts_curv(j)
               temp(index)=xyz(k,j,m)
               index=index+1
            end do
```

```
        end do

c Diagnostics on temp array..(i.e. input data)
        if (diag .eq. .true.) then
           write(6,*) ' '
           write(6,*) '========== NEW CURVE ========== '
           write(6,*) 'Data for curve number=',j
           write(6,*) '     k   m  indx   input data'
           index=1
           do m=1,ndep_gpar
              do k=1,npts_curv(j)
                write(6,fmt='(1x,3i4,f12.6)') k,m,index,
     &                                  temp(index)
                index=index+1
              end do
           end do
        end if
c End Diagnostics

c ***
c Set up dtgpar arguments
c ***
        npt_gpar(1) = npts_curv(j) !Number of data points in s direction
        ndom_gpar   = 1             !Number of indep variables (i.e. s,t)
c        ndep_gpar   = 1             !Number of depend variables (i.e. x,y,z)
        ndim_gpar   = npts_curv(j)  !Max parameter dimension
c
        call dtgpar (npt_gpar,temp,ndom_gpar,ndep_gpar,work,nwork,
     &                 ndim_gpar,t_gpar,ier)
        do l=1,npt_gpar(1)
           ll=npts+l
           sdir(ll)=t_gpar(l)
        enddo
        npts=npt_gpar(1)+npts
        if (ier .ne. 0) then
           write(6,*) ' DTGPAR returned IER=',ier
           stop
        end if

c Diagnostics
        if (diag .eq. .true.) then
           write(6,*) ' '
           write(6,*) '*** DIAG. ON T_GPAR ***'
c          write(6,fmt='(1x,'(1) Curve number=',i3)') j
           do loop2=1,ndim_gpar
              write(6,fmt='(1x,i6,2f12.6)')
     &         loop2,t_gpar(loop2)
           end do
        end if
c End Diagnos

c*************************************************************************
c Fit input curve pts using DTNSI
c*************************************************************************
```

```
         do ii=1,ndep
            if (ii. eq. 1) then
               icc=-1
            else
               icc=2
            end if

c Load input array of dependant variables
            do kk=1,npts_curv(j)
               temp(kk)=xyz(kk,j,ii)
               write(6,*) ' dtnsi ',kk,ii,temp(kk)
            end do
            k=ndeg+1
            ncoef=(npts_curv(j)-2)+k
            nc=5+(ndep+1)*ncoef+k
            call dtnsi(npts_curv(j),
     &                 t_gpar,
     &                 temp,ndeg,icc,hold,nwork,maxc,
     &                 c_carray,nc,ier)
         end do
         do ll=1,100
           write(6,*) ' c_array ',ll,c_carray(ll)
         enddo

c Diagnostics
         if (diag .eq. .true.) then
            call fit_diag(c_carray,j)
         end if
c Error Check
         if (ier .lt. 0) then
             write(6,*) ' DTNSI returned IER=',ier
             stop
         end if

c*********************************************************************
c Generate points at constant parametric locations using max number of
c points contained in any one input curve...load back into data base
c*********************************************************************
         call dtsepp(c_carray,maxpts,work,nwork,temp,ier)

c ***
c Reload surf data base with new refitted data
c ***
         npts_curv(j)=maxpts

         incr=1
         do ii=1,ndep
            do kk=1, maxpts
              xyz(kk,j,ii)=temp(incr)
               write(6,*) ' new temp ',ii,kk,incr,xyz(kk,j,ii)
              incr=incr+1
            end do
         end do
```

```
c***********************************************************************
c Generate parametric array based on the same knot set for each curve
c***********************************************************************

         do i=1,maxpts
           t_gpar(i)=
     &              real((i-1)/(maxpts-1.0))
         end do

c Diagnostics
         if (diag .eq. .true.) then
            write(6,*) ' '
            write(6,*) '*** DIAG. ON T_GPAR ***'
c           write(6,fmt='(1x,' Curve number=',i3)') j
            do loop2=1,maxpts
               write(6,fmt='(1x,i6,f12.6)')
     &         loop2,t_gpar(loop2)
            end do
         end if
c End Diagnostics

c***********************************************************************
c Fit new interpolated input points using DTNSI
c***********************************************************************
         do ii=1,ndep
            if (ii. eq. 1) then
               icc=-1
            else
               icc=2
            end if

c Load input array of dependant variables into temp array
            do kk=1,npts_curv(j)
               temp(kk)=xyz(kk,j,ii)
            end do

            call dtnsi(npts_curv(j),
     &                 t_gpar,
     &                 temp,ndeg,icc,hold,nwork,maxc,
     &                 c_carray,
     &                 len_c,ier)

c Error Check
            if (ier .lt. 0) then
               write(6,*) ' DTNSI returned IER=',ier
               stop
            end if

c Check size of C array against what is output
            call dtcsiz(c_carray,isize,ier)
            if (ier .ne. 0) then
               write(6,*) ' DTCSIZ returned IER=',ier
            end if
            if (isize .ne. len_c) then
```

```
            write(6,*) 'C size does not match!!'
            stop
         end if

      end do

c Diagnostics
      if (diag .eq. .true.) then
         call fit_diag(c_carray,j)
      end if
c End Diagnostics


c ***
c Load curve data into tmp array
c ***

      do nloop=1,len_c
         c_tmp(ipntr_s(j) + nloop-1) =
     &         c_carray(nloop)
         ich=ipntr_s(j)+nloop-1
      end do

c Set pointer into tmp for each location of next curve
      if (j .ne. ncurv) then
         ipntr_s(j+1) = len_c + ipntr_s(j)
      end if
c***
   end do        !end curve loop ...end of surface(i)


c ***
c Store first point of each section to parameterize t space
c ***


c Use first point of station...pnt(1)
   index=1
   do m=1,ndep_gpar
      do i=1,ncurv
         temp(index)=xyz(1,i,m)
         write(6,*) ' t temp ',i,m,temp(index)
         index=index+1
      end do
   end do

c Diagnostics on temp array..(i.e. input data)
   if (diag .eq. .true.) then
      write(6,*) ' '
      write(6,*) ' '
      write(6,*) '========== T CURVE ========== '
c     write(6,fmt='(1x,'Data for T parameter =')')
      write(6,*) '     k    m    indx    input data'
      index=1
```

```
         do m=1,ndep_gpar
            do k=1,ncurv
              write(6,fmt='(1x,3i4,f12.6)') k,m,index,
     &                                   temp(index)
              index=index+1
            end do
         end do
      end if
c End Diagnostics


c Set up dtgpar arguments
      npt_gpar(1) = ncurv         !Number of data points in t direction
      ndom_gpar   = 1                      !Number of indep variables (i.e.
s,t)
c      ndep_gpar   = 1             !Number of depend variables (i.e. x,y,z)
      ndim_gpar   = ncurv         !Max parameter dimension

      call dtgpar (npt_gpar,temp,ndom_gpar,ndep_gpar,work,nwork,
     &       ndim_gpar,s_gpar,ier)
      do l=1,npt_gpar(1)
         tdir(l)=s_gpar(l)
         write(6,*) ' tdir ',l,tdir(l)
      enddo
      if (ier .ne. 0) then
         write(6,*) ' DTGPAR returned IER=',ier
         stop
      end if

c Diagnostics
      if (diag .eq. .true.) then
         write(6,*) ' '
         write(6,*) '*** DIAG. ON NEW T_GPAR ***'
c        write(6,fmt='(1x,'(1) Curve number=',i3)') j
         do loop2=1,ndim_gpar
            write(6,fmt='(1x,i6,2f12.6)')
     &      loop2,s_gpar(loop2)
         end do

c Check on new c_tmp array
         write(6,*) ' '
         write(6,*) ' '
         write(6,*) ' **** C_TMP **** '
         call fit_diag(c_tmp,999)

      end if


c ********************************************************************
c Fit surface to curves
c ********************************************************************

      call dtcrbl(c_tmp,len_c,
```

```
     &             s_gpar,
     &             ncurv, ndeg, work, nwork,
     &             s_carray, ier)
      do i=1,100
        write(6,*) ' s_carray ',i,s_carray(i)
      enddo
      if (ier .ne. 0) then
         write(6,*) ' DTCRBL returned IER=',ier
         if ((ier .eq. 1) .or. (ier .eq. 2)) then
            write(6,*) ' ier=1 => some illconditioning'
            write(6,*) ' ier=2 => severe illconditioning'
         else
            stop
         endif
      end if

c Diagnostics
      if (diag .eq. .true.) then
         call fit_diag(s_carray,id)
      end if

c Determine length of C array
      call dtcsiz(s_carray,len_c,ier)
      if (ier .ne. 0) then
         write(6,*) ' DTCSIZ returned IER=',ier
      end if

c
      return
      end
      subroutine fit_diag(carray,id)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(ndim=6,ndep=6)
      parameter(maxc=(ndim+1)*imxs*jmxs)
      dimension carray(maxc)

      double precision plo(2),phi(2)
      integer ploc(3),qloc(4)
      integer kord(2),ncoef(2)

      call dtget(carray,.true.,2,n,mraw,mdep,kord,ncoef,plo,phi,ier)

      write(6,*) ' '
      write(6,*) ' *** DTGET ON C ARRAY ***'
      write(6,fmt='(1x,/,
     &      ''(1) Curve number=                              '',i3,/,
     &      ''(2) No. indep. variables=                      '',i3,/,
     &      ''(3) No. dep. variables c(2)=                   '',i3,/,
     &      ''(4) No. dep. variables {c(2)-1 if c(2) neg.}='',i3,/,
     &      ''(5) Order of spline U and V=                   '',2i5,/,
     &      ''(6) No. of basis funtions in U and V=          '',2i10,/,
     &      ''(7) IER error flag=                            '',i10,/,
     &      ''(8) Parameter limit low=                       '',2f12.6,/,
```

```
     &       ''(9) Parameter limit high=                        '',2f12.6)')
     &         id,n,mraw,mdep,kord,ncoef,ier,plo,phi
      return
      end
c      ssssssssssssssssssssssssssssssssssss
      subroutine load(mode,idir,nspts,ft)
c      ssssssssssssssssssssssssssssssssssss

      implicit double precision (a-h,o-z)
      include 'csdcfd.par'

        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg

      dimension ft(dnstr+3)

c
        m=0
        do k=1,ksg
          do j=1,jsg
            do i=1,isg
              m=m+1
              if(idir.eq.1) ft(m)=sxs(i,j,k)
              if(idir.eq.2) ft(m)=sys(i,j,k)
              if(idir.eq.3) ft(m)=szs(i,j,k)
            enddo
          enddo
        enddo
c
      return
      end
      subroutine search(ipdir,lout,mout,iout,jout,xa,ya,za,xs,ys,zs)

      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c
c      xs,ys,zs     - structural grid points
c      sxs,sys,szs - structural mode shapes values at grid points
c      xa,ya,za     - aerodynamic grid points
c      sxa,sya,sza - calculated aerodynamic deflections
c      isg,jsg,ksg - number of structural points
c      is,js,ks     - number of structural points ( for each mode)
c      ia,ja,ka     - number of aerodynamic points
c
c
        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg
       common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       dimension xs(4),ys(4),zs(4)

       iout=0
       jout=0
```

```
      lout=0
      mout=0

      go to (300,200,100) ipdir
c
c           x-y plane (ipdir=3)
c
 100  continue
      do l=1,jsg-1
      do i=1,isg-1
        m=(l-1)*isg+i
        mp1=m+isg
c        write(6,*) ' xy ',xt(m),xa,xt(m+1)
        if(xt(m).gt.xa) go to 150
        if(xt(m).le.xa.and.xt(m+1).ge.xa) then
            if(yt(m).gt.ya.and.yt(m+1).gt.ya) go to 120
            if(yt(m).le.ya.and.yt(mp1).ge.ya) then
                mout=m
                jout=l
                iout=i
                lout=1
                go to 500
            endif
            if(yt(m+1).le.ya.and.yt(mp1+1).ge.ya) then
                mout=m
                jout=l
                iout=i
                lout=1
                go to 500
            endif
        endif
 150    continue
      enddo
 120        continue
      enddo
c        compute the extrapolation locations
      lout=0
      mout=0
      m=1
      if(xt(m).gt.xa.or.xt(m+isg-1).lt.xa) then
       if(xt(m).gt.xa) iout=1
       if(xt(m+isg-1).lt.xa) iout=isg
       if(yt(iout).gt.ya) then
            jout=1
            lout=-3
             return
       endif
       do j=1,jsg-1
           m=(j-1)*isg+iout
           if(yt(m).le.ya.and.yt(m+isg).ge.ya) then
                   lout=-1
                   jout=j
                   return
           endif
```

```
          enddo
         if (yt((jsg-1)*isg+iout).lt.ya) then
                jout=jsg
                lout=-3
                return

         endif
        endif
c                   outside span
        m=1
        if(yt(m).gt.ya.or.yt((jsg-1)*isg).lt.ya) then
         if(yt(m).gt.ya) jout=1
         if(yt((jsg-1)*isg+1).lt.ya) jout=jsg
         if(xt(m).gt.(xa)) then
              iout=1
              lout=-3
              return
         endif
         if(xt((jsg-1)*isg+isg).lt.xa) then
              iout=isg
              lout=-3
              return
         endif
         do i=1,isg-1
             m=(jout-1)*isg+i
             if(xt(m).le.xa.and.xt(m+1).ge.xa) then
                     lout=-2
                     iout=i
                     return
             endif
         enddo
        endif
        write(6,*) 'error in search', xa, ya
        return
c             y-z plane (ipdir=2)
  200   continue
        do l=1,jsg-1
        do i=1,isg-1
          m=(l-1)*isg+i
          mp1=m+isg
          if(zt(m).gt.za) go to 250
          if(zt(m).le.za.and.zt(m+1).ge.za) then
            iout=i
              if(yt(m).gt.ya) go to 220
              if(yt(m).le.ya.and.yt(mp1).ge.ya) then
                  jout=1
                  mout=m
                  lout=1
                  go to 500
              endif
  220         continue
          endif
  250     continue
        enddo
```

```
      enddo
      write(6,*) 'error ',ya,za
      lout=0
      return
c           x-z plane (ipdir=1)
  300 continue
      do l=1,jsg-1
      do i=1,isg-1
        m=(l-1)*isg+i
        mp1=m+isg
        if(xt(m).gt.xa) go to 350
        if(xt(m).le.xa.and.xt(m+1).ge.xa) then
            if(zt(m).gt.za) go to 320
            if(zt(m).le.za.and.zt(mp1).ge.za) then
                jout=l
                iout=i
                mout=m
                lout=1
                go to 500
            endif
  320       continue
        endif
  350   continue
      enddo
      enddo
      write(6,*) 'error ',xa,za
      lout=0
      return
c             load structural panel
  500 continue
      jp=mout+isg
      jp1=mout+isg+1
      xs(1)=xt(mout)
      ys(1)=yt(mout)
      zs(1)=zt(mout)
      xs(2)=xt(mout+1)
      ys(2)=yt(mout+1)
      zs(2)=zt(mout+1)
      xs(4)=xt(jp)
      ys(4)=yt(jp)
      zs(4)=zt(jp)
      xs(3)=xt(jp1)
      ys(3)=yt(jp1)
      zs(3)=zt(jp1)
c
c             final checks
c
     if(ipdir.eq.2) go to 600
     if(xa.ge.xs(4).and.xa.le.xs(3)) go to 600
     xr=(xs(4)-xs(1))*(ya-ys(1))/(ys(4)-ys(1))
     if(xr.le.xa) go to 600
     xr=(xs(3)-xs(2))*(ya-ys(2))/(ys(3)-ys(2))
     if(xr.ge.xa) go to 600
     if(iout.eq.1) then
```

```
           lout=0
           return
       endif
      mout=mout-1
      go to 500
600   continue
      if(ipdir.eq.1) go to 700
      if(ya.ge.ys(1).and.ya.ge.ys(2)) go to 700
      yr=(ys(2)-ys(1))*(xa-xs(1))/(xs(2)-xs(1))
      if(yr.le.ya) go to 700
      yr=(ys(3)-ys(4))*(xa-xs(4))/(xs(3)-xs(4))
      if(yr.ge.ya) go to 700
      if(jout.eq.1) then
           lout=0
           return
       endif
      mout=mout-isg
      go to 500
700   continue
      if(ipdir.eq.3) go to 800
      if(za.ge.zs(4).and.za.le.zs(3)) go to 800
      zr=(zs(4)-zs(1))*(xa-xs(1))/(xs(4)-xs(1))
      if(zr.le.za) go to 800
      zr=(zs(3)-zs(2))*(xa-xs(2))/(xs(3)-xs(2))
      if(zr.ge.za) go to 800
      if(iout.eq.1) then
           lout=0
           return
       endif
      mout=mout-1
      go to 500

800   return
      end
c
      subroutine bivarin(xd,yd,fd,x,y,f,ierr)

c  Programmer: V. M. Kaladi, Nov 2, '93

      implicit double precision (a-h,o-z)

      parameter (eps=1e-5, itmax=500, eps2=1e-3)
      dimension xd(4),yd(4),fd(4)
      dimension a(2,2)

c Given the values of a function f(x,y) at four points
c ((xd(i),yd(i), i=1,4) as fd(i), i=1,4 and the point
c (x,y) inside the quadrilateral formed by the data points,
c this subroutine computes the linear bivariate interpolated value
c of f(x,y).

c statement functions
c g(u,v)= sum_(j=1,4) of gj*psij(u,v)
c        = (g0 + ga*u + gb*v + gc*u*v)
```

```
c where gj are values at the data points.
      g0(g1,g2,g3,g4)= 0.25*( g1 + g2 + g3 + g4)
      ga(g1,g2,g3,g4)= 0.25*(-g1 + g2 + g3 - g4)
      gb(g1,g2,g3,g4)= 0.25*(-g1 - g2 + g3 + g4)
      gc(g1,g2,g3,g4)= 0.25*( g1 - g2 + g3 - g4)
      gd(g1,g2,g3,g4)= ((g1-g2)*(g1-g2)+(g3-g4)*(g3-g4))
c end statement functions
c
c check to see if there are identical points
c
      do l=1,4
        chk = gd(xd(l),x,yd(l),y)
c        chk=(xd(l)-x)*(xd(l)-x)+(yd(l)-y)*(yd(l)-y)
        if(sqrt(chk).le.eps) then
          f=fd(l)
          return
        endif
      enddo

      ierr= 0

      x0= g0(xd(1),xd(2),xd(3),xd(4))
      y0= g0(yd(1),yd(2),yd(3),yd(4))
      xa= ga(xd(1),xd(2),xd(3),xd(4))
      ya= ga(yd(1),yd(2),yd(3),yd(4))
      xb= gb(xd(1),xd(2),xd(3),xd(4))
      yb= gb(yd(1),yd(2),yd(3),yd(4))
      xc= gc(xd(1),xd(2),xd(3),xd(4))
      yc= gc(yd(1),yd(2),yd(3),yd(4))

      up= 0.0
      vp= 0.0
      iter=0
 10   continue
        a(1,1)= xa + xc*vp
        a(1,2)= xb
        a(2,1)= ya
        a(2,2)= yb + yc*up
        if((a(1,1)*a(2,2) - a(2,1)*a(1,2)).eq.0.0) go to 20
        deti= 1.0/(a(1,1)*a(2,2) - a(2,1)*a(1,2))
        u= (a(2,2)*(x-x0) - a(1,2)*(y-y0))*deti
        v= (a(1,1)*(y-y0) - a(2,1)*(x-x0))*deti
        errnorm= dsqrt(  (u-up)**2 + (v-vp)**2 )
        up=u
        vp=v
        iter= iter+1
      if (errnorm .gt. eps .and. iter .lt. itmax) go to 10

      if (iter .ge. itmax) then
        print*,'Error, max no. of iterations exceeded '
        print*,'in subroutine bivarin'
        ierr= 1
        write(6,*) ' errnorm = ',errnorm,' eps = ',eps
        write(6,*) ' points ',xa,ya
```

```
          write(6,*) ' xd ',xd(1),xd(2),xd(3),xd(4)
          write(6,*) ' yd ',yd(1),yd(2),yd(3),yd(4)
          stop
        end if

c        if (u .lt. -1.0 .or. u .gt. 1. .or. v .lt. -1 .or. v .gt. 1) then
c          print*,'Error, interpolation point (x,y) is not within '
c          print*,'the quadrilateral of data points.'
c          err= .true.
c          return
c        end if

c Assuming any point falling outside the quadrilateral of data points
c is due to round off error the point is reset to the nearest boundary.
        if (abs(u).gt. 1.0) u= int(u)
        if (abs(v).gt. 1.0) v= int(v)

        f=      g0(fd(1),fd(2),fd(3),fd(4)) +
     &    u * ga(fd(1),fd(2),fd(3),fd(4)) +
     &    v * gb(fd(1),fd(2),fd(3),fd(4)) +
     &    u*v*gc(fd(1),fd(2),fd(3),fd(4))
c
c
        t = (x -xd(1))/(xd(2)-xd(1))
        u = (y -yd(1))/(yd(4)-yd(1))
        fa= (1.-t)*(1.-u)*fd(1) + t*(1.-u)*fd(2) + t*u*fd(3)
     1     + (1.-t)*u*fd(4)

        return
c             The det is 0., so we missed an identical point the
c                 first time through
   20   continue

c check to see if there are identical points
c
        mchk=0
        chkmin=9.999e5
        do l=1,4
          chk = gd(xd(l),x,yd(l),y)
          if(chk.lt.chkmin) then
               chkmin=chk
               mchk=l
          endif
        enddo
        f=fd(mchk)
        return
        end
C -----------------------------------------------------------------------
        subroutine plane1(x,y,z,n,ipdir,planar)
        implicit double precision (a-h,o-z)
C This routine checks to see if the surface resides within a single plane
C It uses input structural coordinates
C planar = .true.
C        = .false.
```

```
c ------------------------------------------------------------------------
      integer ipdir, planar
      dimension x(n), y(n), z(n)
c
c local variables
c
      dimension a(3), b(3), c(3), d(3)
c
c external functions
c
      double precision fmag
c
c scan thru all points & check cross product
c
      eps=1e-28
      ixycnt=0
      iyzcnt=0
      ixzcnt=0
      do i=1, n
     if (i.eq.1) then
        d(1) = x(n-1) - x(i)
        d(2) = y(n-1) - y(i)
        d(3) = z(n-1) - z(i)
          else
        d(1) = x(1) - x(i)
        d(2) = y(1) - y(i)
        d(3) = z(1) - z(i)
          endif
     if (i.eq.n) then
        b(1) = x(2) - x(i)
        b(2) = y(2) - y(i)
        b(3) = z(2) - z(i)
          else
        b(1) = x(n) - x(i)
        b(2) = y(n) - y(i)
        b(3) = z(n) - z(i)
     endif

     call acrossb(d,b,a)       ! find unit normal
c
c Division by zero taking place in this section
c was a(c)=a(c)/fmag(a,3)
c
     a(1) = a(1)/(fmag(a,3)+eps)
     a(2) = a(2)/(fmag(a,3)+eps)
     a(3) = a(3)/(fmag(a,3)+eps)
c
c check to see if points reside on x-y plane
c check if axis and unit normal correspond
c
     b(1) = 0    ! axis normal
     b(2) = 0
     b(3) = 1
     call acrossb(a,b,c)
```

```
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixycnt = ixycnt + 1
c
c check to see if points reside on y-z plane
c
      b(1) = 1     ! axis normal
      b(2) = 0
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) iyzcnt = iyzcnt + 1
c
c check to see if points reside on x-z plane
c
      b(1) = 0     ! axis normal
      b(2) = 1
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixzcnt = ixzcnt + 1
       enddo
c
c set direction
c
       icnt = 0
       if (ixycnt.eq.n) then
      ipdir = 3 ! surface resides in x-y plane
      planar = 1
      icnt = icnt + 1
       elseif (ixzcnt.eq.n) then
      ipdir = 2 ! surface resides in x-z plane
      planar = 1
      icnt = icnt + 1
       elseif (iyzcnt.eq.n) then
      ipdir = 1 ! surface resides in y-z plane
      planar = 1
      icnt = icnt + 1
       else
      planar = 0
       endif
c
c check checksum
c
       if (icnt.gt.1) then
      write(6,*) 'ERROR: In PLANE routine'
      write(6,*) 'Possible grid problem'
      stop
       endif
       return
       end
c ----------------------------------------------------------
c This subroutine finds c = a X b
c a,b,c are 3D vectors
       subroutine acrossb(a,b,c)
```

```
      implicit double precision (a-h,o-z)
      double precision a(3), b(3), c(3)

      c(1) = a(2)*b(3) - b(2)*a(3)
      c(2) = b(1)*a(3) - a(1)*b(3)
      c(3) = a(1)*b(2) - b(1)*a(2)

      return
      end
c -------------------------------------------------------
      double precision function fmag(array,num)
      integer num
      double precision array(num),sum

      sum = 0.0
      do i=1, num
     sum = sum + array(i)*array(i)
      enddo

      fmag = dsqrt(sum)
      return
      end
c ---------------------------------------------------------
      subroutine polint(xa,ya,n,x,y,dy)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(nmax=dnstr)
      dimension xa(nmax),ya(nmax),c(nmax),d(nmax)
      ns=1
      dif=abs(x-xa(1))
      do i=1,n
         write(101,*) ' polint ',i,xa(i),ya(i)
      enddo
      do i=1,n
        dift=abs(x-xa(i))
        if(dift.lt.dif) then
           ns=i
           dif=dift
        endif
        c(i)=ya(i)
        d(i)=ya(i)
      enddo
      y=ya(ns)
      ns=ns-1
      do m=1,n-1
        do i=1,n-m
          ho=xa(i)-x
          hp=xa(i+m)-x
          w=c(i+1)-d(i)
          den=ho-hp
          if(den.eq.0.) then
             write(6,*) 'polint problem '
             write(6,*)  x
             do ii=1,n
```

```
              write(6,*) ii,xa(ii),ya(ii)
              enddo
              stop
           endif
           den=w/den
           d(i)=hp*den
           c(i)=ho*den
         enddo
         if(2*ns.lt.n-m) then
              dy=c(ns+1)
         else
            dy=d(ns)
            ns=ns-1
         endif
         y=y+dy
        enddo
        return
        end
c
      subroutine plane(ipdir,planar)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ x(dnstr),y(dnstr),z(dnstr),ft(dnstr+3),nspts
      integer ipdir, planar
      dimension a(3), b(3), c(3), d(3)
C -----------------------------------------------------------------------
C This routine checks to see if the surface resides within a single plane
C It uses input structural coordinates
C planar = .true.
C        = .false.
C -----------------------------------------------------------------------

c
c external functions
c
      double precision fmag
c
c scan thru all points & check cross product
c
      eps=1e-28
      ixycnt=0
      ixzcnt=0
      iyzcnt=0
      n=nspts
      do i=1, n
     if (i.eq.1) then
        d(1) = x(n-1) - x(i)
        d(2) = y(n-1) - y(i)
        d(3) = z(n-1) - z(i)
          else
        d(1) = x(1) - x(i)
        d(2) = y(1) - y(i)
        d(3) = z(1) - z(i)
          endif
```

```
      if (i.eq.n) then
         b(1) = x(2) - x(i)
         b(2) = y(2) - y(i)
         b(3) = z(2) - z(i)
          else
         b(1) = x(n) - x(i)
         b(2) = y(n) - y(i)
         b(3) = z(n) - z(i)
      endif

      call acrossb(d,b,a)       ! find unit normal
c
c Division by zero taking place in this section
c was a(c)=a(c)/fmag(a,3)
c
      a(1) = a(1)/(fmag(a,3)+eps)
      a(2) = a(2)/(fmag(a,3)+eps)
      a(3) = a(3)/(fmag(a,3)+eps)
c
c check to see if points reside on x-y plane
c check if axis and unit normal correspond
c
      b(1) = 0     ! axis normal
      b(2) = 0
      b(3) = 1
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixycnt = ixycnt + 1
c
c check to see if points reside on y-z plane
c
      b(1) = 1     ! axis normal
      b(2) = 0
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) iyzcnt = iyzcnt + 1
c
c check to see if points reside on x-z plane
c
      b(1) = 0     ! axis normal
      b(2) = 1
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixzcnt = ixzcnt + 1
        enddo
c
c set direction
c
        icnt = 0
        write (6,*) ' Plane counts are ',ixycnt,ixzcnt,iyzcnt
        if (ixycnt.eq.n) then
      ipdir = 3 ! surface resides in x-y plane
```

```
      planar = 1
      icnt = icnt + 1
       elseif (ixzcnt.eq.n) then
      ipdir = 2 ! surface resides in x-z plane
      planar = 1
      icnt = icnt + 1
       elseif (iyzcnt.eq.n) then
      ipdir = 1 ! surface resides in y-z plane
      planar = 1
      icnt = icnt + 1
       else
      planar = 0
c             if plane is curved, then the counts won't be equal
c             to n.  But, the other counts should also be equal
c             to 0.  Correct for this
         if(ixycnt.gt.0.and.ixzcnt.eq.0.and.iyzcnt.eq.0) then
           planar=1
           ipdir=3
           icnt=icnt+1
         endif
         if(ixycnt.eq.0.and.ixzcnt.gt.0.and.iyzcnt.eq.0) then
           planar=1
           ipdir=2
           icnt=icnt+1
         endif
         if(ixycnt.eq.0.and.ixzcnt.eq.0.and.iyzcnt.gt.0) then
           planar=1
           ipdir=3
           icnt=icnt+1
         endif
       endif
c
c check checksum
c
       if (icnt.gt.1) then
      write(6,*) 'ERROR: In PLANE routine'
      write(6,*) 'Possible grid problem'
      stop
       endif
       return
       end
```

18.

APPENDIX C - ALGORITHM CODES

```
csdcfd.par:
c user-defined parameters
c       imxs = streamwise (x-direction) points on structural grid
c       jmxs = spanwise (y-direction) points on structural grid
c       kmxs = normal (z-direction) points on structural grid
c          *** kmxs=1 --- always!
c       imxa = streamwise (x-direction) points on aerodynamic grid
c       jmxa = spanwise (y-direction) points on aerodynamic grid
c       kmxa = normal (z-direction) points on aerodynamic grid
c          *** kmxa=1 --- always!
c       nwk = work array for nubs (dt_nurbs)
c             30k appears to be fine for all cases examined
c             DT_NURBS will flag if it's not high enough
        parameter (imxs=20,jmxs=30,kmxs=1)
        parameter (imxa=220,jmxa=80,kmxa=1)
        parameter (nwk=30000)
c  computed parameters
c       dnstr = total number of structural data points on surface
c       dnaro = total number of aerodynamic data points on surface
        integer dnstr,dnaro
        parameter (dnaro=imxa*jmxa*kmxa,dnstr=imxs*jmxs*kmxs)
c
*********************************************************************
c
c   This program handles the 4 3-D methods
c
         program main
         implicit double precision (a-h,o-z)
         include 'csdcfd.par'


         external cputim
c
c       xs,ys,zs     - structural grid points
c       sxs,sys,szs - structural mode shapes values at grid points
c       xa,yz,za     - aerodynamic grid points
c       sxa,sya,sza - calculated aerodynamic deflections
c       isg,jsg,ksg - number of structural points
c       is,js,ks     - number of structural points ( for each mode)
c       ia,ja,ka     - number of aerodynamic points
c
c
        common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
        common /str/  xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
        common/filen/fnm
         common /struct/ sxs(imxs,jmxs,kmxs)
      &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
      &,js,ks,isg,jsg,ksg
```

```
       dimension xs(imxs,jmxs,kmxs),ys(imxs,jmxs,kmxs),
     1  zs(imxs,jmxs,kmxs)

        dimension f(dnaro,3)
c --- device dependent        real oldtim,time,cputim
        character*80 fnm,fnm1,fnm2,fnm3,fnm4
c
c  Read in the file names containing the input data
c
111     continue
c
c  Clear all of the variables
c
        npts=0
        nspts=0
        do l=1,dnaro
         xa(l)=0.0
         ya(l)=0.0
         za(l)=0.0
         fa(l)=0.0
         f(l,3)=0.0
        enddo
        do l=1,dnstr
         xt(l)=0.0
         yt(l)=0.0
         zt(l)=0.0
         ft(l)=0.0
        enddo
        ft(dnstr+1)=0.0
        ft(dnstr+2)=0.0
        ft(dnstr+3)=0.0
c
c  Read in the new filename information
c
c*** -- device dependent        oldtim=0.0
        write(*,*)'Enter the structural grid filename'
        read(*,'(A)',end=9000)fnm1
        write(*,*)'Enter the structural data filename'
        read(*,'(A)')fnm2
        write(*,*)'Enter the aerodynamic grid filename'
        read(*,'(A)')fnm3

    open(unit=13,file=fnm1,status='unknown')
    open(unit=15,file=fnm2,status='unknown')
    open(unit=17,file=fnm3,status='unknown')

c
c read in data pertaining to structural grid and data
c

    read(13,*)isg,jsg,ksg
        if(isg.eq.0) isg=1
        if(jsg.eq.0) jsg=1
        if(ksg.eq.0) ksg=1
```

```
         nspts=isg*jsg*ksg
c
c            Error check on dimensions
c
         if(nspts.gt.dnstr) then
            write(6,*) ' You have exceeded the dnstr dimension ',
     1       nspts
            stop
         endif
         if(isg.gt.imxs) then
            write(6,*) ' You have exceeded the imxs dimension ',
     1       isg
            stop
         endif
         if(jsg.gt.jmxs) then
            write(6,*) ' You have exceeded the jmxs dimension ',
     1       jsg
            stop
         endif
         if(ksg.gt.kmxs) then
            write(6,*) ' You have exceeded the kmxs dimension ',
     1       ksg
            stop
         endif
c             End error checking
c
c             Read remainder of structural file
c
         read(13,*) (xt(m),m=1,nspts),
     &     (yt(m),m=1,nspts),(zt(m),m=1,nspts)
         close(13)

      read(15,*)ns
         read(15,*)is,js,ks
         if(is.eq.0) is=1
         if(js.eq.0) js=1
         if(ks.eq.0) ks=1
         if(is.ne.isg.or.js.ne.jsg.or.ks.ne.ksg) then
            write(6,*) ' Grid and Data Dimensions Dont Match'
            write(6,*) ' I : ',isg,is
            write(6,*) ' J : ',jsg,js
            write(6,*) ' K : ',ksg,ks
            stop
          endif
       read(15,*)
     &    (((sxs(i,j,k),i=1,is),j=1,js),k=1,ks),
     &    (((sys(i,j,k),i=1,is),j=1,js),k=1,ks),
     &    (((szs(i,j,k),i=1,is),j=1,js),k=1,ks)
          close(15)
c
c read in data pertaining to aerodynamic grid and data
c

      read(17,*)ia,ja,ka
```

```
      if(ia.eq.0) ia=1
      if(ja.eq.0) ja=1
      if(ka.eq.0) ka=1
      npts=ia*ja*ka
c
c         Error check on dimensions
c
      if(npts.gt.dnaro) then
         write(6,*) ' You have exceeded the dnaro dimension ',
     1    npts
         stop
      endif
      if(ia.gt.imxa) then
         write(6,*) ' You have exceeded the imxa dimension ',
     1    ia
         stop
      endif
      if(ja.gt.jmxa) then
         write(6,*) ' You have exceeded the jmxa dimension ',
     1    ja
         stop
      endif
      if(ka.gt.kmxa) then
         write(6,*) ' You have exceeded the kmxa dimension ',
     1    ka
         stop
      endif
c           End error checking
c
c           Finish reading aerodynamic data
        read(17,*) (xa(m),m=1,npts),(ya(m),m=1,npts),
     &  (za(m),m=1,npts)
      close(17)
c
c         fill output arrays with zeros
c
      do m=1,3
      do l=1,npts
      f(l,m)=0.0
      enddo
      enddo
c
c Prompt user for which scheme to use for interpolation
c
 112  continue
      write(*,1)
      write(*,2)
      write(*,4)
      write(*,5)
      write(*,6)
      read(*,*) meth
c       assume that only one mode shape per file!
      mode=1
      if(meth.lt.0.or.meth.gt.4) go to 112
```

```
c
c Call subroutine of chosen method
c
        if(meth .eq. 1) then
c***--device dependent        stime=cputim(oldtim)
c           Infinite Plate Spline
c               (Must choose what components here)
c
          write(*,7)
          read(*,*) ncomp
          if(ncomp.lt.4) then
             nss=ncomp
             nee=ncomp
          else
             nss=1
             nee=3
          endif
          do l=nss,nee
          call load(mode,l,nspts,ft)
          call ips(mode,ierr)
          do m=1,npts
            f(m,l)=fa(m)
          enddo
c             redefine aerodynamic data (modified in ips)
      open(unit=17,file=fnm3,status='unknown')
      read(17,*)ia,ja,ka
          read(17,*) (xa(m),m=1,npts),(ya(m),m=1,npts),
     &    (za(m),m=1,npts)
        close(17)
          enddo
        elseif(meth .eq. 2)then
c                   Thin Plate Spline
          call mq(3,f)
        elseif(meth .eq. 3)then
c                   Multiquadrics
          call mq(1,f)
        elseif(meth .eq. 4)then
c                   NUBS
          call nurbs(mode,f)
        endif
c
c  Check amount of required cpu runtime for this method
c
c*** -- device dependent      time=cputim(stime)

      call output(mode,ia,ja,ka,f)

      close(13)
      close(15)
      close(17)

c*** -- device dependent       time=abs(time-stime)
       call calcmax(mode,f)
c*** -- device dependent       oldtime=time
```

```
      go to 111
9000   continue
c
c          . format statements
c
 1    format(///4x,'Enter choice of interpolation methods:')
 2    format(7x,'1]  Infinite Plate Splines')
 4    format(7x,'2]  Thin Plate Spline')
 5    format(7x,'3]  Multi-Quadrics')
 6    format(7x,'4]  NUBS')
 7    format(///4x,
     1' What kind of data is to be interpolated?',/,
     17x,' 1]  x-component only ',/,
     27x,' 2]  y-component only ',/,
     37x,' 3]  z-component only ',/,
     47x,' 4]  all three components ')
c
      stop
      end
c
c          This subroutine generates the output files
c
      subroutine output(mode,ia,ja,ka,f)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts

       dimension f(dnaro,3)
       common/filen/fnm

      character*80  ofile,nfile,mfile,lfile,fnm

       write(6,*) ' Enter the output file type : '
       write(6,*) ' 0 : Plot3d , 1 : SGF '
       read(*,*) itype
       write(6,*)' Enter aerodynamic grid + mode shape file name'
       read(*,'(A)')nfile
       write(6,*)' Enter interpolated function file name'
       write(6,*)' Must end in character x '
       read(*,'(A)')ofile
 10     continue

       nvar=1
      if(itype.eq.0) then
       open(unit=102,file=nfile,status='unknown')
       write(102,*)ia,ja,ka
       write(102,*)((xa(m)+f(m,1)),m=1,npts),
     &              ((ya(m)+f(m,2)),m=1,npts),
     &              ((za(m)+f(m,3)),m=1,npts)
       write(103,*)ia,ja,ka
       write(103,*)((f(m,1)),m=1,npts),
     &              ((f(m,2)),m=1,npts),
     &              ((f(m,3)),m=1,npts)
       close(102)
```

```
          close(103)
          go to 1000
c
          open(unit=103,file=ofile,status='unknown')
          write(103,*)ia,ja,ka,nvar
          write(103,*)(f(ll,1),ll=1,npts)
          close(103)
c
          do ll=1,80
            if(ofile(ll:ll).eq.'x') ie=ll
          enddo
          ofile(1:ie)=ofile(1:ie-1)//'y'
          open(unit=103,file=ofile,status='unknown')
          write(103,*)ia,ja,ka,nvar
          write(103,*)(f(ll,2),ll=1,npts)
          close(103)
c
          do ll=1,80
            if(ofile(ll:ll).eq.'y') ie=ll
          enddo
          ofile(1:ie)=ofile(1:ie-1)//'z'
          open(unit=103,file=ofile,status='unknown')
          write(103,*)ia,ja,ka,nvar
          write(103,*)(f(ll,3),ll=1,npts)
          close(103)
         else
          open(unit=102,file=nfile,status='unknown')
          nz=1
          nr=0
          write(102,'(A)') nfile
          write(102,*) nz
          write(102,*)ia,ja,ka
          write(102,'(A)') nfile
          write(102,*) nr
          do k=1,ka
          do j=1,ja
          iss=(j-1)*ia+1
          iee=j*ia
          write(102,*)(xa(m)+f(m,1),m=iss,iee)
          write(102,*)(ya(m)+f(m,2),m=iss,iee)
          write(102,*)(za(m)+f(m,3),m=iss,iee)
          enddo
          enddo
          close(102)
          do l=1,3
          if(l.eq.2) then
             do ll=1,80
               if(ofile(ll:ll).eq.'x') ie=ll
             enddo
             ofile(1:ie)=ofile(1:ie-1)//'y'
          else if (l.gt.2) then
             do ll=1,80
               if(ofile(ll:ll).eq.'y') ie=ll
             enddo
```

```
            ofile(1:ie)=ofile(1:ie-1)//'z'
         endif
         open(unit=103,file=ofile,status='unknown')
         do k=1,ka
         do j=1,ja
         iss=(j-1)*ia+1
         iee=j*ia
         write(103,*)(f(m,1),m=iss,iee)
         enddo
         enddo
         close(103)
         enddo
        endif
c
1000    continue
        return
        end
c
c           This subroutine calculates the max/min locations and
c                 magnitudes
c           (From MPROC2D)
        subroutine calcmax(mode,f)
        implicit double precision (a-h,o-z)
        include 'csdcfd.par'

c
c        xs,ys,zs      - structural grid points
c        sxs,sys,szs   - structural mode shapes values at grid points
c        xa,yz,za      - aerodynamic grid points
c        sxa,sya,sza   - calculated aerodynamic deflections
c        isg,jsg,ksg   - number of structural points
c        is,js,ks      - number of structural points ( for each mode)
c        ia,ja,ka      - number of aerodynamic points
c
c
          common /struct/ sxs(imxs,jmxs,kmxs)
       1,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
       1,js,ks,isg,jsg,ksg
         common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
         common /str/  xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
         dimension f(dnaro,3)
C
C  Add to the existing file 7 the check data!
     open(unit=7,file='accuracy.dat',status='unknown')
c  141    read(7,*,end=142)
c         go to 141
c  142    continue
C First, find maximum in structural data
          tmp = dsqrt(sxs(1,1,1)*sxs(1,1,1)
      1   + sys(1,1,1)*sys(1,1,1)
      1   + szs(1,1,1)*szs(1,1,1))
          tmp2 = tmp
          tmp3 = 0
          locmax=1
```

```
        locmin=1
        m=0
        do k=1,ksg
        do j=1,jsg
        do i=1,isg
        m=m+1
        chktmp =dsqrt(sxs(i,j,k)*sxs(i,j,k)
    1    +sys(i,j,k)*sys(i,j,k)
    1    +szs(i,j,k)*szs(i,j,k))
        tmp3 = tmp3 + chktmp
        if (chktmp.gt.tmp) then
           tmp = chktmp
           locmax = m
        endif
        if(chktmp.lt.tmp2) then
           tmp2 = chktmp
           locmin =m
        endif
     enddo
     enddo
     enddo
     pmax = tmp
     pmin = tmp2
     pavg = tmp3/nspts

C Find maximum values in interpolated data
        tmp = dsqrt(f(1,1)*f(1,1)+f(1,2)*f(1,2)+f(1,3)*f(1,3))
        tmp2 = tmp
        tmp3 = 0.0
        min=0
        max=0

        do 10 m=1,npts
           chktmp = dsqrt(f(m,1)*f(m,1)+f(m,2)*f(m,2)
    >            +f(m,3)*f(m,3))
           tmp3 = tmp3 + chktmp
           if (chktmp.gt.tmp) then
              tmp = chktmp
              max = m
           endif
           if(chktmp.lt.tmp2) then
              tmp2 = chktmp
              min = m
           endif
10      continue
       dmax = tmp
       dmin = tmp2
       davg = tmp3/npts

C Print max values
    if (pmax.eq.0.0) then
           pdiff = 100*(dmax-pmax)
        else
           pdiff = 100*(dmax-pmax)/pmax
```

```
         endif

         write(6,8) pmax,xt(locmax),yt(locmax),zt(locmax)
         write(6,9) dmax,xa(max),ya(max),za(max)
         write(6,13) dmax-pmax,pdiff
         write(7,8) pmax,xt(locmax),yt(locmax),zt(locmax)
         write(7,9) dmax,xa(max),ya(max),za(max)
         write(7,13) dmax-pmax,pdiff

C Print min values
     if (pmin.eq.0.0) then
            pdiff = 100*(dmin-pmin)
         else
            pdiff = 100*(dmin-pmin)/pmin
         endif
         write(6,11) pmin,xt(locmin),yt(locmin),zt(locmin)
         write(6,12) dmin,xa(min),ya(min),za(min)
         write(6,13) dmin-pmin,pdiff
         write(7,11) pmin,xt(locmin),yt(locmin),zt(locmin)
         write(7,12) dmin,xa(min),ya(min),za(min)
         write(7,13) dmin-pmin,pdiff
   8     format( ' The true maximum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
   9     format( ' The interpolated maximum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
  11     format( ' The true minimum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
  12     format( ' The interpolated minimum is ',g10.4,
     1 ' at (x,y,z) :',3(g10.4,','))
  13     format( ' The difference is ',g13.5,'(',g9.3,'%)')
      return
      end
      subroutine ips(mode,ierr)
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c    This subroutine solves the infinite plate spline equations
c       for a three-dimensional surface
c    The equations in this code were extracted from a code developed
c       by Ram Shan at GTRI in 1993 named MPROC3D. This code was a
c       3-D extension of a NASA-Langley code called MPROC, provided
c       by Beth Rausch in the Unsteady Aerodynamics Branch.
c
c
c    The code assumes that each mode is called independently for
c       each direction (up to 3 directions).
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c
      dimension singular(3),nd(3),emax(3)
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
      common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
      common /surf/ xb(dnstr),yb(dnstr)
```

```
      common /work/ array(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), dum(dnstr+3)

c
      data eps/1.e-28/
      data singular/0.,0.,0./

      do l=1,dnstr+3
       w(l)=0.0
       xb(l)=0.0
       yb(l)=0.0
       dum(l)=0.0
       do ll=1,dnstr+3
         array(l,ll)=0.0
         v(l,ll)=0.0
       enddo
      enddo


c      singular(1)  = 5.0e-4
c      singular(2)  = 5.0e-4
c      singular(3)  = 1.0e-4
c      singular(1)  = 1.0e-3
c      singular(2)  = 1.0e-3
c      singular(3)  = 5.0e-4

c     ------------------------------------------------
c     check to see if surface is planar
c     ------------------------------------------------
c
      call plane(ipdir,itmp)
      if(ipdir.lt.1.or.ipdir.gt.3) then
          write(6,*) ' Error in plane subroutine '
          write(6,*) ' Plane returned is ',ipdir
          stop
      endif

c     ------------------------------------------------
c     ipdir = 1 x-y plane splinal matrix data
c     ipdir = 2 x-z plane splinal matrix data
c     ipdir = 3 y-z plane splinal matrix data
c     ------------------------------------------------
      if (itmp.eq.1) then
     if (ipdir.eq.1) then
c           ipdir=1 is the x-y plane, data to update is z (i2=3)
        i2s = 3
        i2e = 3
         endif
     if (ipdir.eq.2) then
c           ipdir=2 is the x-z plane, data to update is y (i2=2)
        i2s = 2
        i2e = 2
         endif
     if (ipdir.eq.3) then
```

```
c          ipdir=3 is the y-z plane, data to update is x (i2=1)
       i2s = 1
       i2e = 1
         endif
      else
     i2s = 1
     i2e = 3
      endif


      rbu1 = 0.0
      rbu2 = 0.0
      rbu3 = 0.0
      rbv1 = 0.0
      rbv2 = 0.0
      rbv3 = 0.0

      do 5 i2=i2s,i2e
c
c store appropriate values in x & y arrays in order to use
c existing subroutines
c
      if (i2 .eq. 1) then
     write(6,*) 'Processing x-y coordinate data'
         call fdinert(xt,yt,nspts,thz,rbu3, rbv3,xcg,ycg)
         do ii=1,nspts                        ! use x & y coordinates
            xb(ii) = (xt(ii) - xcg)/rbu3        ! shift coords to cg
            yb(ii) = (yt(ii) - ycg)/rbv3
c
c  Experienced division by zero read: beta=atan2(yb(ii),xb(ii)
c
            beta = atan2(yb(ii), xb(ii)+eps)
            rad = sqrt(xb(ii)**2 + yb(ii)**2)  ! rotate to principle axes
       xb(ii) = rad*cos(beta-thz)
       yb(ii) = rad*sin(beta-thz)
         enddo
      endif
      if (i2 .eq. 2) then
         write(6,*) 'Processing x-z coordinate data'
         call fdinert(xt,zt,nspts,thy,rbu2,rbv2,xcg,zcg)
         do 29 ii=1,nspts                      ! use x & z coordinates
            xb(ii) = (xt(ii) - xcg)/rbu2        ! shift coords to cg
            yb(ii) = (zt(ii) - zcg)/rbv2
c
c  Incase of Division by zero beta was beta=atan2(yb(ii),xb(ii))
c
       beta = atan2(yb(ii), xb(ii)+eps)
            rad = sqrt(xb(ii)**2 + yb(ii)**2)  ! rotate to principle axes
       xb(ii) = rad*cos(beta-thy)
       yb(ii) = rad*sin(beta-thy)
29       continue
      endif
      if (i2 .eq. 3) then
     write(6,*) 'Processing y-z coordinate data'
```

```
         call fdinert(yt,zt,nspts,thx,rbu1,rbv1,ycg,zcg)
         do 30 ii=1,nspts                        ! use y & z coordinates
            xb(ii) = (yt(ii) - ycg)/rbu1          ! shift coords to cg
            yb(ii) = (zt(ii) - zcg)/rbv1
c
c  Division by zero beta was : beta=atan2(yb(ii),xb(ii))
c
        beta = atan2(yb(ii), xb(ii)+eps)
            rad = sqrt(xb(ii)**2 + yb(ii)**2)   ! rotate to principle axes
        xb(ii) = rad*cos(beta-thx)
        yb(ii) = rad*sin(beta-thx)
30       continue
      endif
c
c           fill solution matrix array
c
      do 2 i=1,nspts
         do 1 j=1,i
            r2 = (xb(i)-xb(j))**2+(yb(i)-yb(j))**2
            g= r2 * dlog(r2+eps)
        array(i,j) = g
        array(j,i) = g
    1    continue
    2 continue

      do 3 j=1,nspts
     array(nspts+1,j) = 1.0
     array(j,nspts+1) = 1.0
    3 continue

      array(nspts+1,nspts+1) = 0.0

      do 4 j=1,nspts
     array(nspts+2,j) = xb(j)
     array(j,nspts+2) = xb(j)

    4 continue

      array(nspts+2, nspts+1) = 0.0
      array(nspts+1, nspts+2) = 0.0
      array(nspts+2, nspts+2) = 0.0

      do 6 j=1,nspts
     array(nspts+3,j) = yb(j)
     array(j,nspts+3) = yb(j)
    6 continue

      array(nspts+3, nspts+1) = 0.0
      array(nspts+1, nspts+3) = 0.0
      array(nspts+3, nspts+2) = 0.0
      array(nspts+2, nspts+3) = 0.0
      array(nspts+3, nspts+3) = 0.0

      n1=nspts+3
```

```
c
c      -------------------------------------------------------
c      call routine to decompose main matrix.
c      -------------------------------------------------------
       call svdcmp(n1)
c
c scale structural coordinates & cfd grid coordinates
c rotate coordinates to principle axes
c Select appropriate normal coordinate for interpolation
c
          if (i2.eq.1) then
             write(6,*) 'Calculating xy-plane deflections'
             do m=1,npts
                xa(m) = (xa(m) - xcg)/rbu3
                ya(m) = (ya(m) - ycg)/rbv3
        rad = sqrt(xa(m)**2 + ya(m)**2)
        beta = atan2(ya(m),xa(m)+eps)
                xa(m) = rad*cos(beta-thz)
        ya(m) = rad*sin(beta-thz)
             enddo
          elseif (i2.eq.3) then
             write(6,*) 'Calculating yz-plane deflections'
             do m=1,npts
                ya(m) = (ya(m) - ycg)/rbu1
                za(m) = (za(m) - zcg)/rbv1
        beta = atan2(za(m),ya(m)+eps)
        rad = sqrt(ya(m)**2 + za(m)**2)
        ya(m) = rad*cos(beta-thx)
        za(m) = rad*sin(beta-thx)
             enddo
          else if (i2.eq.2) then
             write(6,*) 'Calculating xz-plane deflections'
             do m=1,npts
                xa(m) = (xa(m) - xcg)/rbu2
               za(m) = (za(m) - zcg)/rbv2
        beta = atan2(za(m), xa(m)+eps)
        rad = sqrt(xa(m)**2 + za(m)**2)
        xa(m) = rad*cos(beta-thy)
        za(m) = rad*sin(beta-thy)
             enddo
          else
             write(6,*) ' Code error no plane identified ',i2
        endif
c
        ft(nspts+1) = 0.
        ft(nspts+2) = 0.
        ft(nspts+3) = 0.
c
c  Ask if user wants to change the threshold
c
        irest=0
        call chthres(nspts,irest,ipdir,singular(ipdir),nd(ipdir),
     1     emax(ipdir))
        nn=nspts+3
```

```
      nnn=dnstr+3
      call slvdriver(nn,nnn,nd(ipdir),singular(ipdir),
     1     emax(ipdir))

c
c     -----------------------------------------
c     compute deflections at collocation points
c     -----------------------------------------

      if(i2.eq.1) then
c             update z deflections
      do m=1,npts
        fa(m)=zfun(xa(m),ya(m),eps)
      enddo
      else if (i2.eq.2) then
c             update y deflections
      do m=1,npts
        fa(m)=zfun(xa(m),za(m),eps)
      enddo
      else if (i2.eq.3) then
c             update x deflections
      do m=1,npts
        fa(m)= zfun(ya(m),za(m),eps)
      enddo
      endif
5     continue
c
      return
      end
c
c     FFFFFFFFFFFFFFFFFFFF
      function zfun(xx,yy,eps)
c     FFFFFFFFFFFFFFFFFFFF
      implicit double precision (a-h,o-z)
     include 'csdcfd.par'
     common /str/ xt(dnstr),yt(dnstr),zt(dnstr),zeta(dnstr+3),nspts
     common /surf/ xi(dnstr),eta(dnstr)
c     ----------------------------------------------------------
c     spline evaluation function for a symmetric surface spline
c     ----------------------------------------------------------
      b1=zeta(nspts+1)
      b2=zeta(nspts+2)
      b3=zeta(nspts+3)
      zfun = b1+b2*xx+b3*yy

      do 1 i=1,nspts
         dx2 = (xx-xi(i))**2
         dy2 = (yy-eta(i))**2
         rl = dx2 + dy2
     zfun = zfun + zeta(i)*rl*dlog(rl+eps)
   1  continue

      return
      end
```

```
c---------------------------------------------------------------
      subroutine slvdriver(n3,mp,nd,singular,emax)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),b(dnstr+3),nspts
      common /work/ u(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), x(dnstr+3)
c
      common /file/ iunit
c
c
c Edit diagonal

      nd = 0.0
      wmax = 0.0
      do i=1, n3
     wmax = max(wmax, w(i))
      enddo
      emax = wmax
      wmin=wmax*singular
      do i=1, n3
    if (w(i) .lt. wmin) then
       w(i) = 0.0
       nd = nd + 1
    endif
      enddo
c
c Call Solver
c
      call svbksb(n3, n3 )
c
c Copy solution matrix into b vector
c
      do i = 1, n3
         b(i) = x(i)
      enddo

      return
      end
C ---------------------------------------------------------------------
      subroutine svbksb(m,n)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ xt(dnstr),yt(dnstr),zt(dnstr),b(dnstr+3),nspts
      common /work/ u(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), x(dnstr+3)
      dimension tmp(dnstr+3)
C ---------------------------------------------------------------------
C This subroutine solves Ax = b where A = U, W, V as returned
C from svdcmp
C m,n = logical dimensions of A
C b = RHS vector
C x = output solution vector
C ---------------------------------------------------------------------
```

```
      if (n.ne.m) n=m
      do j=1,n
         s=0.0
         if (w(j).ne.0.0) then
            do i=1,m
               s=s+u(i,j)*b(i)
            enddo
            s=s/w(j)
         endif
         tmp(j) = s
      enddo
      do j=1,n
         s=0.0
         do jj=1,n
            s=s+v(j,jj)*tmp(jj)
         enddo
         x(j) = s
      enddo
      return
      end
c---------------------------------------------
      subroutine fdinert(xf,yf,n, th, rbu, rbv, xcg, ycg)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c       dimension xf(n), yf(n)
      dimension xf(dnstr), yf(dnstr)
c -----------------------------------------------------------------
c This subroutine finds the moments of inertia for given coordinate info
c The moments of inertia is used to find the principle axes direction
c It assumes that each coordinate represents a particle of mass = 1
c xf, yf - coordinate info
c th - principle axes direction angle (rad)
c rbu - radius of gyration in u-direction
c rbv - radius of gyration in v-direction
c -----------------------------------------------------------------
c
      xi = 0.0
      yi = 0.0
      xyi = 0.0
      qx = 0.0
      qy = 0.0
c
c Calculate 1st & 2nd moments of inertia
c
      do ii= 1, n
         xi = xi + yf(ii)*yf(ii)
         yi = yi + xf(ii)*xf(ii)
         xyi = xyi + xf(ii)*yf(ii)
         qx = qx + yf(ii)
         qy = qy + xf(ii)
      enddo
c
c Calculate cg locations
c
```

```
      xcg = qy / n
      ycg = qx / n
c
c Calculate direction of principle axes & principle moments
c
      tmp = xi - yi
      if (tmp.lt.0.0) then
         th  = .5*atan2(-1.0*xyi, .5*tmp)
         th1 = .5*atan2(xyi, -.5*tmp)
      else
         th  = .5*atan2(-1.0*xyi, .5*tmp)
         th1 = .5*atan2(xyi, -.5*tmp)
      endif
      ui = xi*(cos(th))**2 -2.0*xyi*sin(th)*cos(th) + yi*(sin(th))**2
      vi = xi*(cos(th1))**2 -2.0*xyi*sin(th1)*cos(th1)+yi*(sin(th1))**2
c
c Calculate radii of gyration
c
      rbu = sqrt(ui/n)
      rbv = sqrt(vi/n)

      return
      end
*-----------------------------------------------------------------
      subroutine svdcmp(m)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /work/ a(dnstr+3,dnstr+3),v(dnstr+3,dnstr+3),
     1  w(dnstr+3), dum(dnstr+3)
      dimension rv1(dnstr+3)
*-----------------------------------------------------------------
* This subroutine performs a Singular Value Decomposition (SVD)
* a = u * w * (v)-transpose
* It is obtained from
*   "Numerical Recipes", By  Press & Flannery et al.,
*      Cambridge University Press, 1989, pgs. 57- 64
* a = matrix logical dimensions m by n
*     physical dimensions mp by np
* u = stored in a on output
* w = diagonal matrix of singular values
* v = NOT stored as transpose on output
* if n > m, fill A to make square with zero rows
*-----------------------------------------------------------------

      n=m
c
c
      if (m.lt.n) then
     write(6,*) 'Error:Inside SVDCMP routine'
     write(6,*) 'Augment A matrix with zero rows'
     stop
      endif

*
```

```
* Householder reduction to bidiagonal form
*

         g = 0.0
         scale = 0.0
         anorm = 0.0
         do 25 i=1,n
        l=i+1
        rv1(i)=scale*g
        g=0.0
        s=0.0
           scale = 0.0
        if (i.le.m) then
           do k=i,m
              scale=scale+abs(a(k,i))
               enddo
           if (scale.ne.0.0) then
              do k=i,m
             a(k,i) = a(k,i)/scale
             s=s+a(k,i)*a(k,i)
                   enddo
              f=a(i,i)
              g=-sign(sqrt(s),f)
              h=f*g-s
              a(i,i)=f-g
              if (i.ne.n) then
             do j=l,n
                s=0.0
                do k=i,m
               s=s+a(k,i)*a(k,j)
                         enddo
                f=s/h
                do k=i,m
               a(k,j) = a(k,j)+f*a(k,i)
                         enddo
                      enddo
                   endif
              do k=i,m
             a(k,i) = scale*a(k,i)
                   enddo
               endif
            endif
        w(i) = scale * g
        g=0.0
        s=0.0
        scale=0.0
        if((i.le.m).and.(i.ne.n)) then
           do k=l,m
              scale=scale+abs(a(i,k))
               enddo
           if (scale.ne.0.0) then
              do k=l,n
             a(i,k) = a(i,k)/scale
             s=s+a(i,k)*a(i,k)
```

```
            enddo
            f=a(i,l)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,l) = f - g
            do k=l,n
          rv1(k) = a(i,k)/h
            enddo
            if(i.ne.m) then
           do j=l,m
              s=0.0
              do k=l,n
             s=s+a(j,k)*a(i,k)
              enddo
              do k=l,n
             a(j,k) = a(j,k)+s*rv1(k)
              enddo
           enddo
            endif
            do k=l,n
           a(i,k) = scale*a(i,k)
            enddo
         endif
          endif
      anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25    continue
*
* Accumulation of right hand side transformations
*
      do 32 i=n,1,-1
         if (i.lt.n) then
            if (g.ne.0.0) then
                do j=l,n
                   v(j,i)=(a(i,j)/a(i,l))/g
                 enddo
            do j=l,n
               s=0.0
               do k=l,n
                  s=s+a(i,k)*v(k,j)
               enddo
               do k=l,n
                  v(k,j)=v(k,j)+s*v(k,i)
               enddo
            enddo
            endif
            do j=l,n
               v(i,j) = 0.0
               v(j,i) = 0.0
            enddo
         endif
         v(i,i) = 1.0
         g = rv1(i)
         l = i
32    continue
```

```
*
* Accumulation of Left Hand Side transformations
*
      do 39 i=n,1,-1
         l=i+1
         g=w(i)
         if (i.lt.n) then
            do j = l,n
              a(i,j)=0.0
            enddo
         endif
         if (g.ne.0.0) then
            g = 1.0/g
            if (i.ne.n) then
               do j=l,n
                  s=0.0
                  do k=l,m
                     s=s+a(k,i)*a(k,j)
                  enddo
                  f=(s/a(i,i))*g
                  do k=i,m
                     a(k,j)=a(k,j)+f*a(k,i)
                  enddo
               enddo
            endif
            do j=i,m
               a(j,i)=a(j,i)*g
            enddo
         else
            do j=i,m
               a(j,i) = 0.0
            enddo
         endif
         a(i,i) = a(i,i) + 1.0
39    continue
*
* Diagonalization of bidiagonal form
*
      do 49 k=n,1,-1       ! singular values loop
      do 48 its=1,30    ! iteration loop

         do l=k,1,-1    ! test for splitting
            nm=l-1      ! rv1(1) always zero
            if ((abs(rv1(l))+anorm).eq.anorm) goto 2
            if ((abs(w(nm))+anorm).eq.anorm) goto 1
         enddo
1        c=0.0          ! cancellation of rv1(l), if l>1
         s=1.0
         do i=l,k
            f = s*rv1(i)
            rv1(i) = c*rv1(i)
            if ((abs(f)+anorm).eq.anorm) goto 2
            g = w(i)
            h = sqrt(f*f+g*g)
```

```
            w(i) = h
            h = 1.0/h
            c = (g*h)
            s = -(f*h)
            do j=1,m
               y=a(j,nm)
               z=a(j,i)
               a(j,nm) = (y*c) + (z*s)
               a(j,i)  = -(y*s) + (z*c)
            enddo
         enddo
2        z = w(k)
         if (l.eq.k) then            ! Convergence
            if (z.lt.0.0) then       ! singular value made > 0
               w(k) = -z
               do j=1,n
                 v(j,k) = -v(j,k)
               enddo
            endif
            goto 3
         endif
         if (its.eq.30) then
            write(6,*) 'Error: In SVDCMP routine'
            write(6,*) 'No convergence in 30 iterations'
            write(6,*) 'Matrix cannot be inverted!'
            stop
         endif
         x = w(l)
         nm = k-1
         y = w(nm)
         g = rv1(nm)
         h = rv1(k)
         f = ((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y)
         g = sqrt(f*f  + 1.0)
         f = ((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x

c Next QR transformation

         c = 1.0
         s = 1.0
         do j=l,nm
            i=j+1
            g=rv1(i)
            y=w(i)
            h=s*g
            g=c*g
            z=sqrt(f*f+h*h)
            rv1(j) = z
            c = f/z
            s = h/z
            f = (x*c)+(g*s)
            g = -(x*s) + (g*c)
            h = y*s
            y = y*c
```

```
            do jj=1,n
                x = v(jj,j)
                z = v(jj,i)
                v(jj,j) = (x*c)+(z*s)
                v(jj,i) = -(x*s)+(z*c)
            enddo
            z = sqrt(f*f + h*h)
            w(j) = z
            if (z.ne.0.0) then
               z = 1.0/z
               c = f*z
               s = h*z
            endif
            f = (c*g) + (s*y)
            x = -(s*g) + (c*y)
            do jj = 1, m
               y = a(jj,j)
               z = a(jj,i)
               a(jj,j) = (y*c) + (z*s)
               a(jj,i) = -(y*s) + (z*c)
            enddo
         enddo
         rv1(l) = 0.0
         rv1(k) = f
         w(k) = x
48       continue
3        continue
49    continue
      return
      end
C ---------------------------------------------------------------
      subroutine chthres(nspts,irest,idir,singular,nd,emax)
       implicit double precision (a-h,o-z)
       character*5 plane(3)
       real tmp

       plane(1) = ' y-z '
       plane(2) = ' x-z '
       plane(3) = ' x-y '

       write(6,*) 'Do you want to change the threshold?(y/n)'
       write(6,103)
       if ((ans.eq.'y').or.(ans.eq.'Y')) then
       write(6,*)
       write(6,*) '-------------------'
       write(6,*) 'CHANGING THRESHOLDS'
       write(6,*) '-------------------'
       write(6,100) 'plane', 'threshold','values deleted', 'max eigen'
       write(6,100) '-----', '---------','--------------', '---------'
       write(6,101) plane(idir), singular, nd, nspts, emax
       write(6,*)
      irest = 1
 10       continue
      write(6,104) 'Enter threshold:'
```

```
      write(6,103)
      read(5,*) tmp
      write(6,*)
      write(6,*) 'You have entered:'
      write (6,105) 'Threshold:',tmp
      write(6,*) 'Is that correct(y/n)?'
      read(5,'(a)') ans
      if ((ans.eq.'y').or.(ans.eq.'Y')) then
        singular = tmp
          goto 99
            else
          goto 10
            endif
       endif
c
c            format statements
c
100    format (1x, a5, 3x, a9, 3x, a14, 3x, a9)
101    format (1x, a5, 3x, e8.1, 3x, '(',i3,'/',i3,')', 3x, f9.3)
103    format ('--->',$)
104    format(a,1x,i)
105    format(a,1x,i,e8.1)
c
99    continue
      ans='n'
      return
       end
c
c***********************************************************************
c
c                          SUBROUTINE MQ
c
c                 Multiquadric-Biharmonic Method
c
c     Created:        SEP06,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine mq(naux,fo)
c
      implicit none
c
      integer           mqpt,NWK
      integer           imxs,jmxs,kmxs,nmxs
      integer           imxa,jmxa,kmxa,nmxa
c
      include 'csdcfd.par'
      include 'mq.par'
c
      integer           naux
      double precision fi(dnstr,3),fo(dnaro,3)
c
      double precision rmin,rmax
```

```
c
      integer          npi,npo
      double precision xi,yi,zi,ft
      double precision xo(dnaro),yo(dnaro),zo(dnaro),fa(dnaro)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     str common contains the structural surface data
c
c     nspts = total number of points on the surface
c     xi(dnstr) = Cartesian X location of data points
c     yi(dnstr) = Cartesian Y location of data points
c     zi(dnstr) = Cartesian Z location of data points
c     ft(dnstr) = Local data to be interfaced
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      common /str/ xi(dnstr),yi(dnstr),zi(dnstr),ft(dnstr+3),npi
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     aero common contains the aerodynamic (CFD) surface data
c
c     npts = total number of points on the surface
c     xo(dnaro) = Cartesian X location of data points
c     yo(dnaro) = Cartesian Y location of data points
c     zo(dnaro) = Cartesian Z location of data points
c     fa(dnaro) = Local data to be interfaced
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      common /aero/ xo,yo,zo,fa,npo
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c     struct common contains the structural mode data
c
c     sxs(imxs,jmxs,kmxs) = Cartesian X location of data points
c     sys(imxs,jmxs,kmxs) = Cartesian Y location of data points
c     szs(imxs,jmxs,kmxs) = Cartesian Z location of data points
c     isg = total number of points along the x-direction
c     jsg = total number of points along the y-direction
c     ksg = total number of points along the z-direction
c     is  = points along the x-direction
c     js  = points along the y-direction
c     ks  = points along the z-direction
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      integer          is,js,ks,isg,jsg,ksg
      double precision sxs(imxs,jmxs,kmxs),sys(imxs,jmxs,kmxs)
      double precision szs(imxs,jmxs,kmxs)
c
      common /struct/sxs
     &,sys,szs
     &,is,js,ks,isg,jsg,ksg
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
c.... Local variables
c
      integer          i,j,k,m
      integer          ni,no
      integer          ninterv
      integer          idiv,jdiv,kdiv
      integer          nscale
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c.... Dynamic Memory (real and integer in two separated spaces)
c
      integer          nr1,nr2,nr3,nr4,nr5,nr6,nr7,nr8
      integer          nr9,nr10,nr11,nr12,nr13,nr14,nr15
      integer          ni1,ni2,ni3,ni4
      integer          nexti,nextr,nmaxiv,nmaxrv
      integer          ivect(nnmaxi)
      double precision rvect(nnmaxr)
c
      common/point/    nexti,nextr,nmaxiv,nmaxrv
      common/memoi/    ivect
      common/memor/    rvect
c
      nmaxiv= nnmaxi
      nmaxrv= nnmaxr
c
      open(unit=27,file='rpar')
      read(27,*) rmin,rmax
      read(27,*) nscale
      close(27)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c.... Basic parameters calculation
c
c...... Total number of input (structural) points ---> (npi)
c
c...... Total number of output (aerodynamic) points ---> (npo)
c
c...... Maximum number of points within a sub-region
c
      ni = npi
      no = npo
c
c.... Estimate the number of subdivisions necessary in each direction
c
      idiv = isg/imax + 1
      jdiv = jsg/jmax + 1
      kdiv = ksg/kmax + 1
c
c.... and the total number of sub-regions
c
      ninterv = idiv*jdiv*kdiv
```

```
c
c.... Memory allocation
c
c...... Real part
c
c      nr1  ... xsi(ni,ninterv)
c      nr2  ... ysi(ni,ninterv)
c      nr3  ... zsi(ni,ninterv)
c      nr4  ... fsi(ni,3,ninterv)
c      nr5  ... xso(no,ninterv)
c      nr6  ... yso(no,ninterv,)
c      nr7  ... zso(no,ninterv,)
c      nr8  ... fso(no,3,ninterv)
c
c      nr9  ... xi2(npi)
c      nr10 ... yi2(npi)
c      nr11 ... zi2(npi)
c      nr12 ... xo2(npo)
c      nr13 ... yo2(npo)
c      nr14 ... zo2(npo)
c
       nr1   = 1
       nr2   = nr1 + ni*ninterv
       nr3   = nr2 + ni*ninterv
       nr4   = nr3 + ni*ninterv
       nr5   = nr4 + 3*ni*ninterv
       nr6   = nr5 + no*ninterv
       nr7   = nr6 + no*ninterv
       nr8   = nr7 + no*ninterv
       nextr = nr8 + 3*no*ninterv
c
       nr9   = nextr
       nr10  = nr9  + npi
       nr11  = nr10 + npi
       nr12  = nr11 + npi
       nr13  = nr12 + npo
       nr14  = nr13 + npo
       nr15  = nr14 + npo
c
c...... Integer part
c
c      ni1  ... conecti(npi)
c      ni2  ... conecto(npo,17)
c      ni3  ... npis(ninterv)
c      ni4  ... npos(ninterv)
c
       ni1   = 1
       ni2   = ni1 + npi
       ni3   = ni2 + 17*npo
       ni4   = ni3 + ninterv
       nexti = ni4 + ninterv
c
       write(*,*) 'Last allocated integer =',nexti
c
```

```
      write(*,*)'**********************************************'
      write(*,*)'PARAMETERS FOR THIS PROBLEM'
      write(*,*)
      write(*,*)'npi    npo       ',npi,npo
      write(*,*)'ni     no        ',ni,no
      write(*,*)'dnstr  dnaro     ',dnstr,dnaro
      write(*,*)'idiv   jdiv      ',idiv,jdiv
      write(*,*)'isg    jsg   ksg ',isg,jsg,ksg
      write(*,*)'is     js    ks  ',is ,js ,ks
      write(*,*)'imax   jmax  kmax',imax,jmax,kmax
      write(*,*)'idiv   jdiv  kdiv',idiv,jdiv,kdiv
      write(*,*)'nexti  nextr     ',nexti,nextr
      write(*,*)'nnmaxi nnmaxr    ',nnmaxi,nnmaxr
      write(*,*)'ninterv          ',ninterv
      write(*,*)
      write(*,*)'**********************************************'
      write(*,*)
c
      if(nmaxrv.lt.nr15.or.nmaxiv.lt.nexti) then
        write(*,*) 'NOT ENOUGH MEMORY FOR THIS CASE'
        write(*,*) 'Last allocated integer position    =',nexti
        write(*,*) 'Maximum allocated integer position =',nmaxiv
        write(*,*) 'Last allocated real position    =',nr15
        write(*,*) 'Maximum allocated real position =',nmaxrv
        if(nmaxrv.lt.nr15 ) STOP 'More real memory is necessary'
        if(nmaxiv.lt.nexti) STOP 'More integer memory is necessary'
      end if
c
c.... Put the input modes in a working format
c
      m=0
      do k=1,ksg
        do j=1,jsg
          do i=1,isg
            m=m+1
            fi(m,1) = sxs(i,j,k)
            fi(m,2) = sys(i,j,k)
            fi(m,3) = szs(i,j,k)
          enddo
        enddo
      enddo
c
c.... Transfer the working space and constants to the MQ solver
c
      call mq1 (dnaro,dnstr,npi,npo,ni,no,ninterv,idiv,jdiv,kdiv,
     1          rmin,rmax,naux,nscale,
     2          ivect(ni1),ivect(ni2),ivect(ni3),ivect(ni4),
     3          xi,yi,zi,fi,xo,yo,zo,fo,
     4          rvect(nr9), rvect(nr10),rvect(nr11),
     5          rvect(nr12),rvect(nr13),rvect(nr14),
     6          rvect(nr1),rvect(nr2),rvect(nr3),rvect(nr4),rvect(nr5),
     7          rvect(nr6),rvect(nr7),rvect(nr8) )
c
c.... END OF SUBROUTINE
```

```
c
      write(*,*)'END'
      write(*,*)
c
      return
      end
c
c*********************************************************************
c
c                      SUBROUTINE MQ1
c
c     Multiquadric-Biharmonic Method - Solution Routine
c
c     Created:       SEP07,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine mq1 (dnaro,dnstr,npi,npo,ni,no,ninterv,idiv,jdiv,kdiv,
     1                rmin,rmax,naux,nscale,
     2                conecti,conecto,npis,npos,
     3                xi0,yi0,zi0,fi,xo0,yo0,zo0,fo,
     4                xi,yi,zi,xo,yo,zo,
     5                xsi,ysi,zsi,fsi,xso,yso,zso,fso)
c
      implicit none
c
      integer          dnaro,dnstr
      integer          npi,npo,ni,no,ninterv,naux,nscale
      integer          idiv,jdiv,kdiv
      integer          conecti(npi),npis(ninterv)
      integer          conecto(npo,9),npos(ninterv)
      double precision rmin,rmax
      double precision xi0(dnstr),yi0(dnstr),zi0(dnstr)
      double precision xo0(dnaro),yo0(dnaro),zo0(dnaro)
      double precision xi(npi),yi(npi),zi(npi),fi(dnstr,3)
      double precision xo(npo),yo(npo),zo(npo),fo(dnaro,3)
      double precision xsi(ni,ninterv),ysi(ni,ninterv)
      double precision zsi(ni,ninterv),fsi(ni,3,ninterv)
      double precision xso(no,ninterv),yso(no,ninterv)
      double precision zso(no,ninterv),fso(no,3,ninterv)
c
c.... Local variables
c
      integer          i,interv
      integer          nr1,nr2,nr3,nr4,nr5,nr6,nr7,nr8,nr9
      integer          ni1
      integer          maxni,maxno,maxn
c
      integer          nexti,nextr,nnmaxi,nnmaxr
      integer          ivect(1)
      double precision rvect(1)
c
      common/point/    nexti,nextr,nnmaxi,nnmaxr
```

```
      common/memor/    rvect
      common/memoi/    ivect
c
c.... Put the input/output grids in a working format
c
      do i=1,npi
        xi(i) = xi0(i)
        yi(i) = yi0(i)
        zi(i) = zi0(i)
      end do
c
      do i=1,npo
        xo(i) = xo0(i)
        yo(i) = yo0(i)
        zo(i) = zo0(i)
      end do
c
c.... Scale the domain to a unity cube [0,1] x [0,1] x [0,1]
c
      if(nscale.eq.1) then
      write(*,*)'Scale the domain to a unity cube ...'
      write(*,*)
      end if
      call scalesr (nscale,npi,npo,naux,
     1              xi,yi,zi,xo,yo,zo)
c
c.... Partition the given domain
c
      write(*,*)'Partition the given domain (x-y-z plane) ...'
      write(*,*)
      call partit (idiv,jdiv,kdiv,npi,ni,npo,no,
     1             ninterv,dnstr,dnaro,
     2             xi,yi,zi,fi,xo,yo,zo,
     3             xsi,ysi,zsi,fsi,xso,yso,zso,
     4             conecti,conecto,npis,npos)
c
c.... Local memory allocation (real part)
c
      maxni = 0
      maxno = 0
      do i=1,ninterv
        write(*,*)'interv  npis(interv)',i,npis(i)
        write(*,*)'interv  npos(interv)',i,npos(i)
        if(npis(i).gt.maxni) maxni = npis(i)
        if(npos(i).gt.maxno) maxno = npos(i)
      end do
      if(maxni.gt.maxno) then
         maxn = maxni
      else
         maxn = maxno
      end if
c
c     nr1   ... B(maxni,maxni)
c     nr2   ... wkarear(maxn)
```

```
c     nr3  ... BIG(maxno,maxni)
c     nr5  ... R(maxni,naux)
c     nr7  ... RI(maxn,naux)
c     nr9  ... alpha(maxni)
c
      nr1   = nextr
      nr2   = nr1 + maxni*maxni
      nr3   = nr2 + maxn
      nr5   = nr3 + maxno*maxni
      nr7   = nr5 + maxni*naux
      nr9   = nr7 + maxn*naux
      nextr = nr9 + maxni
c
c.... Local memory allocation (integer part)
c
c     ni1  ... wkareai(maxn)
c
      ni1   = nexti
      nexti = ni1 + maxn
c
      if(nnmaxr.lt.nextr.or.nnmaxi.lt.nexti) then
        write(*,*) 'NOT ENOUGH MEMORY FOR THIS CASE'
        write(*,*) 'Last allocated integer position    =',nexti
        write(*,*) 'Maximum allocated integer position =',nnmaxi
        write(*,*) 'Last allocated real position    =',nextr
        write(*,*) 'Maximum allocated real position =',nnmaxr
        if(nnmaxr.lt.nextr) STOP 'More real memory is necessary'
        if(nnmaxi.lt.nexti) STOP 'More integer memory is necessary'
      end if
c
c.... Perform the MQ/TPS interpolation in each sub-region
c
      write(*,*)'Perform the interpolation in each sub-region ...'
      write(*,*)
      do interv = 1, ninterv
c
c...... Non-dimensionalization of the coordinates for a
c       given sub-region
c
        if(nscale.eq.1) then
        write(*,*)'Scaling sub-region', interv,' ...'
        write(*,*)
        end if
        call scalesr ( nscale,npis(interv),npos(interv),naux,
     1                 xsi(1,interv),ysi(1,interv),zsi(1,interv),
     2                 xso(1,interv),yso(1,interv),zso(1,interv) )
c
c...... Evaluate the interpolated function (fso) for the sub-region
c
        write(*,*)'Interpolating for sub-region', interv,' ...'
        write(*,*)
c
        call inter (maxni,maxno,maxn,naux,npis(interv),npos(interv),
     1              ni,no,rmin,rmax,
```

```
     2                xsi(1,interv),ysi(1,interv),zsi(1,interv),
     3                xso(1,interv),yso(1,interv),zso(1,interv),
     4                rvect(nr1),rvect(nr5),
     5                rvect(nr2),ivect(ni1),rvect(nr3),rvect(nr7),
     6                rvect(nr9),fsi(1,1,interv),
     7                fso(1,1,interv) )
c
      end do
c
c.... Average the interpolated function over the overlaped areas
c
      write(*,*)'Average over the overlaped areas ...'
      write(*,*)
      call overlap (dnaro,npo,no,ninterv,fso,conecto,
     1              fo)
c
c.... END OF SUBROUTINE
c
      return
      end
c
c
c***********************************************************************
c
c                         SUBROUTINE OVERLAP
c
c     Evaluate the value of the output function that belongs to an
c     overlaping area by simply weighting the interpolated result that
c     comes from the diferent sub-regions
c
c     Created:        SEP08,95
c     Last Modified:
c
c***********************************************************************
c
      subroutine overlap (dnaro,npo,nno,ninterv,fso,conecto,
     1                    fo)
c
      implicit          none
c
      integer           dnaro,npo,nno,ninterv
      integer           conecto(npo,17)
      double precision fso(nno,3,ninterv)
      double precision fo(dnaro,3)
c
      integer           m,i,nint,naux,interv,no
c
      do m=1,npo
        nint = conecto(m,1)
        if(nint.gt.17) STOP 'at overlap - nint > 9'
        do i=1,nint
          naux = 2*i
          interv = conecto(m,naux)
          no = conecto(m,naux+1)
```

```
            fo(m,1) = fo(m,1) + fso(no,1,interv)/nint
            fo(m,2) = fo(m,2) + fso(no,2,interv)/nint
            fo(m,3) = fo(m,3) + fso(no,3,interv)/nint
          end do
        end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*********************************************************************
c
c                        SUBROUTINE SCALESR
c
c     Scales the sub-region to a unity cube [0,1] x [0,1] x [0,1]
c
c     Created:       SEP07,95
c     Last Modified:
c
c*********************************************************************
c
      subroutine scalesr (nscale,npi,npo,naux,
     1                    xsi,ysi,zsi,xso,yso,zso)
c
      implicit         none
c
      integer          nscale,npi,npo,naux
      double precision xsi(npi),ysi(npi),zsi(npi)
      double precision xso(npo),yso(npo),zso(npo)
c
      integer          i
      double precision xmin,xmax,ymin,ymax,zmin,zmax
      double precision auxx,auxy,auxz,dx,dy,dz
c
      double precision eps
      parameter (eps=1.0D-14)
c
c.... Scale the domain to a unit cube ( [0,1] x [0,1] x [0,1] )
c
c...... Find the min. and max. coordinate values along each direction
c       considering all the point involved in the sub-region (both input
c       as well as output points)
c
      xmax = xsi(1)
      xmin = xsi(1)
      ymax = ysi(1)
      ymin = ysi(1)
      zmax = zsi(1)
      zmin = zsi(1)
c
      call mimax (npi,xsi,ysi,zsi,npi,
     1            xmin,xmax,ymin,ymax,zmin,zmax)
c
```

```
      dx = xmax - xmin
      dy = ymax - ymin
      dz = zmax - zmin
c
c.... For the Thin-Plate Spline Method
c     (changes from 2-D to 3-D analysis)
c
c     naux = 2 --> 1-D sub-domain
c          = 3 --> 2-D sub-domain
c          = 4 --> 3-D sub-domain
c
        if(naux.eq.2.or.naux.eq.3.or.naux.eq.4) then
          if(dabs(dx).gt.eps) then
            if(dabs(dy).gt.eps) then
              if(dabs(dz).gt.eps) then
                 naux = 4
              else
                 naux = 3
              end if
            else
              naux = 2
            end if
          end if
        end if
c
c.... If scale is not to take place (nscale.ne.1), return to the
c     calling routine
c
      if(nscale.ne.1) RETURN
c
      call mimax (npo,xso,yso,zso,npo,
     1            xmin,xmax,ymin,ymax,zmin,zmax)
c
c...... All grid points have coordinates between [0,1]
c
c.... Note: If there is no variation along one direction,
c           it is made the constant equal to zero as part
c           of the scaling process
c
      dx = xmax - xmin
      dy = ymax - ymin
      dz = zmax - zmin
c
      if(DABS(dx).gt.eps) then
c
c.... Structural grid along x
c
        do i=1,npi
          xsi(i) = ( xsi(i) - xmin ) / dx
        end do
c
c.... Aerodynamic grid along x
c
        do i=1,npo
```

```
           xso(i) = ( xso(i) - xmin ) / dx
         end do
c
       else
c
c.... Structural grid along x
c
         do i=1,npi
           xsi(i) =  xsi(i) - xmin
         end do
c
c.... Aerodynamic grid along x
c
         do i=1,npo
           xso(i) =  xso(i) - xmin
         end do
c
       end if
c
c----------------------------------------------------------------------
c
       if(DABS(dy).gt.eps) then
c
c.... Structural grid along y
c
         do i=1,npi
           ysi(i) = ( ysi(i) - ymin ) / dy
         end do
c
c.... Aerodynamic grid along y
c
         do i=1,npo
           yso(i) = ( yso(i) - ymin ) / dy
         end do
c
       else
c
c.... Structural grid along y
c
         do i=1,npi
           ysi(i) =  ysi(i) - ymin
         end do
c
c.... Aerodynamic grid along y
c
         do i=1,npo
           yso(i) =  yso(i) - ymin
         end do
c
       end if
c
c----------------------------------------------------------------------
c
       if(DABS(dz).gt.eps) then
```

```
c
c.... Structural grid along z
c
        do i=1,npi
          zsi(i) = ( zsi(i) - zmin ) / dz
        end do
c
c.... Aerodynamic grid along z
c
        do i=1,npo
          zso(i) = ( zso(i) - zmin ) / dz
        end do
c
c.... For the Thin-Plate Spline Method
c     (changes from 2-D to 3-D analysis)
c
      else
c
c.... Structural grid along z
c
        do i=1,npi
          zsi(i) =  zsi(i) - zmin
        end do
c
c.... Aerodynamic grid along z
c
        do i=1,npo
          zso(i) =  zso(i) - zmin
        end do
c
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c************************************************************************
c
c                       SUBROUTINE PARTIT
c
c     Performs the domain partition for the MQ method
c
c     Created:       SEP04,95
c     Last Modified:
c
c************************************************************************
c
      subroutine partit (idiv,jdiv,kdiv,npi,nni,npo,nno,
     1                   ninterv,dnstr,dnaro,
     2                   xi,yi,zi,fi,xo,yo,zo,
     3                   xsi,ysi,zsi,fsi,xso,yso,zso,
     4                   conecti,conecto,npis,npos)
c
```

```
      implicit none
c
      integer          idiv,jdiv,kdiv,npi,nni,npo,nno
      integer          ninterv,dnstr,dnaro
      double precision xi(npi),yi(npi),zi(npi),fi(dnstr,3)
      double precision xo(npo),yo(npo),zo(npo)
c
      integer          npis(ninterv),npos(ninterv)
      integer          conecti(npi),conecto(npo,17)
      double precision xsi(nni,ninterv),ysi(nni,ninterv)
      double precision zsi(nni,ninterv),fsi(nni,3,ninterv)
      double precision xso(nno,ninterv),yso(nno,ninterv)
      double precision zso(nno,ninterv)
c
      integer          ij,ji
c
      integer          i,j,k,m,interv,ni,no,ii,jj
      integer          naux,auxi,auxo
      double precision deltax,deltay,deltaz,tolx,toly,tolz
      double precision xmin,xmax,ymin,ymax,zmin,zmax
      double precision xmin0,ymin0,zmin0
c
c.... Initialize arrays
c
      do i=1,npi
        conecti(i) = 0
      end do
c
      do j=1,9
      do i=1,npo
        conecto(i,j) = 0
      end do
      end do
c
c.... Determine delta_x, delta_y, and delta_z based on the number of
c     subdivisions, as well as the size of the overlaping region based
c     on a percentage of the size of the interval (assumed to be 10%)
c
      xmax  = xi(1)
      xmin0 = xi(1)
      ymax  = yi(1)
      ymin0 = yi(1)
      zmax  = zi(1)
      zmin0 = zi(1)
c
      call mimax (npi,xi,yi,zi,npi,
     1            xmin0,xmax,ymin0,ymax,zmin0,zmax)
c
      call mimax (npo,xo,yo,zo,npo,
     1            xmin0,xmax,ymin0,ymax,zmin0,zmax)
c
      deltax = (xmax - xmin0)/dfloat(idiv)
      deltay = (ymax - ymin0)/dfloat(jdiv)
      deltaz = (zmax - zmin0)/dfloat(kdiv)
```

```
c
c..... This can only be used when one has a unit cube
c       deltax = 1.0/dfloat(idiv)
c       deltay = 1.0/dfloat(jdiv)
c       deltaz = 1.0/dfloat(kdiv)
c
      tolx = 0.1*deltax
      toly = 0.1*deltay
      tolz = 0.1*deltaz
c
c.... Find the association between a grid point and the given region
c
      interv = 0
c
      do k=1,kdiv
       zmin = zmin0 + (k-1)*deltaz - tolz
       zmax = zmin + deltaz + 2.0*tolz
c
       do j=1,jdiv
        ymin = ymin0 + (j-1)*deltay - toly
        ymax = ymin + deltay + 2.0*toly
c
        do i=1,idiv
        interv = interv + 1
        xmin = xmin0 + (i-1)*deltax - tolx
        xmax = xmin + deltax + 2.0*tolx
c
c...... Check which points are in the given region and assemble new
c       arrays for the coordinates
c
c... Check limits of the sub-region
c
        ni = 0
        do m=1,npi
          if(xi(m).ge.xmin.and.xi(m).le.xmax.and.
     1       yi(m).ge.ymin.and.yi(m).le.ymax.and.
     2       zi(m).ge.zmin.and.zi(m).le.zmax) then
            ni = ni + 1
            xsi(ni,interv) = xi(m)
            ysi(ni,interv) = yi(m)
            zsi(ni,interv) = zi(m)
            fsi(ni,1,interv) = fi(m,1)
            fsi(ni,2,interv) = fi(m,2)
            fsi(ni,3,interv) = fi(m,3)
c
c...... Save the connectivity information for the point
            naux = conecti(m)
            if(naux.gt.7) then
              write(*,*)'ERROR in point assignment to a sub-region'
              STOP 'Error-conecti'
            end if
            conecti(m) = naux + 1
c
          end if
```

```
        end do
        npis(interv) = ni
c
        no = 0
        do m=1,npo
          if(xo(m).ge.xmin.and.xo(m).le.xmax.and.
     1       yo(m).ge.ymin.and.yo(m).le.ymax.and.
     2       zo(m).ge.zmin.and.zo(m).le.zmax) then
            no = no + 1
            xso(no,interv) = xo(m)
            yso(no,interv) = yo(m)
            zso(no,interv) = zo(m)
c
c...... Save the connectivity information for the point
            naux = conecto(m,1)
            if(naux.gt.7) then
              write(*,*)'ERROR in point assignment to a sub-region'
              write(*,*)'point naux',m,naux
              write(*,*)'xo(m)  yo(m)',xo(m),yo(m)
              STOP 'Error-conecto'
            end if
            conecto(m,1) = naux + 1
            conecto(m,2*naux+2) = interv
            conecto(m,2*naux+3) = no
c
          end if
        end do
        npos(interv) = no
c
        end do
       end do
      end do
c
c.... Check if the original estimation of the number of intervals
c     (ninterv) has not been miscalculated
c
      if(interv.gt.ninterv) then
        write(*,*) 'From subroutine PARTIT'
        write(*,*) 'Bad estimation of total number of sub-regions'
        write(*,*) 'Estimated number (ninterv) =',ninterv
        write(*,*) 'Needed number (interv) =',interv
        STOP 'ERROR - ninterv too small'
      end if
c
c.... Check if all the points have been assigned to one of the regions
c
      auxi = 0
      do m=1,npi
        if(conecti(m).eq.0) then
          auxi = auxi + 1
         end if
      end do
c
      auxo = 0
```

```
      do m=1,npo
        if(conecto(m,1).eq.0) then
          auxo = auxo + 1
         end if
      end do
c
      if(auxi.ne.0.or.auxo.ne.0) then
        write(*,*)'ERROR in the domain sub-division'
        write(*,*)'Number of input points left over =', auxi
        write(*,*)'Number of output points left over =', auxo
        write(*,*)'Number of input points included =',npi - auxi
        write(*,*)'Number of output points included =',npo - auxo
        STOP 'domain sub-division'
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*************************************************************************
c
c                        SUBROUTINE INTER
c
c     Performs the interpolation based on the MQ or TPS methods
c
c     naux = 1 --> Multiquadric Method
c          = 2 --> 1-D Thin-Plate Spline Method
c          = 3 --> 2-D Thin-Plate Spline Method
c          = 4 --> 3-D Thin-Plate Spline Method
c
c     Created:       SEP26,95
c     Last Modified:
c
c*************************************************************************
c
      subroutine inter (ni,no,nmax,naux,npi,npo,nni,nno,rmin,rmax,
     1                   xi,yi,zi,xo,yo,zo,
     2                   B,R,wkarear,wkareai,BIG,RI,alpha,fi,
     3                   fo)
c
      implicit           none
c
      integer            ni,no,npi,npo,nni,nno
      integer            nmax,naux
      integer            wkareai(nmax)
      double precision rmin,rmax
      double precision xi(ni),yi(ni),zi(ni)
      double precision xo(no),yo(no),zo(no)
      double precision B(ni,ni),BIG(no,ni)
      double precision R(ni,naux),RI(nmax,naux)
      double precision wkarear(nmax),alpha(ni)
      double precision fi(nni,3)
      double precision fo(nno,3)
```

```
c
      integer          i,j
      double precision d
c
c.... Assembling the coefficient matrix [B]
c
c      write(*,*)'   Assembling the coefficient matrix [B]'
c      write(*,*)
c
      call bmatrx (naux,ni,npi,xi,yi,zi,ni,npi,xi,yi,zi,rmin,rmax,
     1             B)
c
c.... Assembling the coefficient matrix [R]
c
c      write(*,*)'   Assembling the coefficient matrix [R]'
c      write(*,*)
      call Rmatrx (ni,naux,ni,npi,xi,yi,zi,
     1             R)
c
c.... Perform first phase of the LU decomposition on matrix [B]
c
c      write(*,*)'   First phase of LU decomposition on [B]'
c      write(*,*)
      call ludcmpd (B,npi,ni,wkareai,d,wkarear)
c
c.... Do a loop over the modes (not implemented because the mode under
c                                consideration is defined at the driver
c                                level, which is not too efficient for
c                                this method)
c      do imode=1,mode
c
c.... Compute the interpolated mode for each of the components
c     of the given input mode
c
c...... x-component
c
      write(*,*)'   Compute the interpolated mode for x-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1             ni,npi,xi,yi,zi,rmin,rmax,
     2             B,R,wkarear,wkareai,BIG,RI,fi(1,1),alpha,
     3             fo(1,1))
c
c...... y-component
c
      write(*,*)'   Compute the interpolated mode for y-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1             ni,npi,xi,yi,zi,rmin,rmax,
     2             B,R,wkarear,wkareai,BIG,RI,fi(1,2),alpha,
     3             fo(1,2))
c
c...... z-component
c
```

```
      write(*,*)'   Compute the interpolated mode for z-component'
      write(*,*)
      call fitfo (nmax,naux,no,npo,xo,yo,zo,
     1            ni,npi,xi,yi,zi,rmin,rmax,
     2            B,R,wkarear,wkareai,BIG,RI,fi(1,3),alpha,
     3            fo(1,3))
c
c.... END OF SUBROUTINE
c
      return
      end
c
c**********************************************************************
c
c                       SUBROUTINE FITFO
c
c     Performs the interpolation based on the MQ or TPS methods
c
c     Created:        SEP26,95
c     Last Modified:
c
c**********************************************************************
c
      subroutine fitfo (nmax,naux,no,npo,xo,yo,zo,
     1                  ni,npi,xi,yi,zi,rmin,rmax,
     2                  B,R,wkarear,wkareai,BIG,RI,fi,alpha,
     3                  fo)
c
      implicit         none
c
      integer          nmax,naux
      integer          ni,npi,no,npo
      integer          wkareai(nmax)
      double precision rmin,rmax
      double precision xi(ni),yi(ni),zi(ni)
      double precision xo(no),yo(no),zo(no)
      double precision B(ni,ni),BIG(no,ni)
      double precision R(ni,naux),RI(nmax,naux)
      double precision alpha(ni),wkarear(nmax)
      double precision fi(ni)
      double precision fo(no)
c
      integer          i,j
      integer          aux2(16)
      double precision err,err2,d
      double precision beta(4)
      double precision auxf(4),aux1(4,4),aux3(16)
c
c.... Backup {fi} in wkarear
c
      do i=1,npi
        wkarear(i) = fi(i)
      end do
c
```

```
c.... Perform second phase of LU decomposition (backsubstitution)
c
c      write(*,*)'   ... Perform second phase of LU decomposition [B]'
c      write(*,*)
      call lubksbd (B,npi,ni,wkareai,fi)
c
c.... Calculate the constants \alpha and
c                        constraint constants \beta
c
c     beta = ( R^T . (B^{-1}.R) )^{-1} . R^T . fi
c
c.... Backup [R] in [R_I]
c
      do j=1,naux
      do i=1,npi
        RI(i,j) = R(i,j)
      end do
      end do
c
c...... Perform second phase of LU decomposition (backsubstitution)
c
c      write(*,*)'   ... Perform second phase of LU decomposition [R]'
c      write(*,*)
      do i=1,naux
        call lubksbd (B,npi,ni,wkareai,RI(1,i))
      end do
c
      call lsmptd (R,ni,npi,naux,RI,nmax,naux,aux1,4)
c
      call lsmptd (R,ni,npi,naux,fi,ni,1,auxf,4)
c
      if(naux.eq.1) then
c
c...... For Multiquadric Method
        beta(1) = auxf(1) / aux1(1,1)
c
        do i=1,npi
          alpha(i) = fi(i) - RI(i,1)*beta(1)
        end do
c
      elseif(naux.eq.2.or.naux.eq.3.or.naux.eq.4) then
c
c...... For Thin-Plate Spline Method
c        write(*,*)'   ... Perform first phase decomposition for [aux1]'
c        write(*,*)
        call ludcmpd (aux1,naux,4,aux2,d,aux3)
c        write(*,*)'   ... Perform second phase decomposition for [aux1]'
c        write(*,*)
        call lubksbd (aux1,naux,4,aux2,auxf)
c        call lubksbd (aux1,3,4,aux2,auxf)
        do i=1,naux
          beta(i) = auxf(i)
        end do
        call lsmpd (RI,nmax,npi,naux,beta,4,1,alpha,ni)
```

```
c
          do i=1,npi
            alpha(i) = fi(i) - alpha(i)
          end do
c
        end if
c
c.... Check on the constraints
c     (the constraint requires that R^T . alpha = 0)
c
      call lsmptd (R,ni,npi,naux,alpha,ni,1,auxf,4)
      write(*,*)'   ... Error in the constraint for alpha'
      do i=1,naux
        write(*,*)'       constraint',i,' = ',auxf(i)
      end do
      write(*,*)
c
c.... Evaluate the coefficient matrix based on both input and output
c     grid points
c
c      write(*,*)'   ... Evaluate [B_IG]'
      call bmatrx (naux,no,npo,xo,yo,zo,ni,npi,xi,yi,zi,rmin,rmax,
     1             BIG)
c
c      write(*,*)'   ... Evaluate [R_I]'
c      write(*,*)
      call Rmatrx (nmax,naux,no,npo,xo,yo,zo,
     1             RI)
c
c.... Evaluate the product [BIG]*{alpha} + [RI]*{beta} = {Hi}
c
c      write(*,*)'   ... Evaluate {Hi}'
c      write(*,*)
      call lsmpd (BIG,no,npo,npi,alpha,ni,1,fo,no)
      call lsmpd (RI,nmax,npo,naux,beta,naux,1,wkarear,nmax)
      do i=1,npo
         fo(i) = fo(i) + wkarear(i)
      end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                      SUBROUTINE BMATRX
c
c     Generates the coefficient matrix based on the basis functions and
c     a user-defined varing parameter "r"
c
c     Created:        SEP04,95
c     Last Modified:
c
```

```
c*********************************************************************
c
      subroutine bmatrx (naux,no,npo,xo,yo,zo,ni,npi,xi,yi,zi,rmin,rmax,
     1                   B)
c
      implicit          none
c
      integer           naux,ni,no,npi,npo
      double precision rmin,rmax
      double precision xo(no),yo(no),zo(no)
      double precision xi(ni),yi(ni),zi(ni)
      double precision B(no,ni)
c
      integer           i,j
      double precision npi1,exp,r2,raux
      double precision xxi,xxj,yyi,yyj,zzi,zzj
c
      double precision eps
      parameter (eps=1.0d-14)
c
c.... Check if MQ or TPS has been selected
c
c---------------------------------------------------------------------
c     Multiquadrics Basis Functions
c---------------------------------------------------------------------
      if(naux.eq.1) then
c
c.... Constant calculation for the varying "r" parameter
c
      npi1 = dfloat(npi) - 1.0
      raux = (rmax/rmin)*(rmax/rmin)
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        yyj = yi(j)
        zzj = zi(j)
        exp = ( dfloat(j) - 1.0 ) / npi1
        r2 = (rmin*rmin)*( (raux)**exp )
        do i=1,npo
          xxi = xo(i)
          yyi = yo(i)
          zzi = zo(i)
          B(i,j) = dsqrt( (xxi - xxj)*(xxi - xxj) +
     1                    (yyi - yyj)*(yyi - yyj) +
     2                    (zzi - zzj)*(zzi - zzj) + r2 )
        end do
      end do
c
      return
      end if
c
c---------------------------------------------------------------------
```

```
c     End of MQ
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c     Thin-Plate Spline Basis Functions
c-----------------------------------------------------------------------
c
c.... 1-D Problems
c
      if(naux.eq.2) then
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        zzj = zi(j)
        do i=1,npo
          xxi = xo(i)
          zzi = zo(i)
          r2 =  (xxi - xxj)*(xxi - xxj) +
     1          (zzi - zzj)*(zzi - zzj)
          if(r2.lt.eps) then
            B(i,j) = 0.0
          else
            B(i,j) = r2 * DLOG(r2) / 2.0
          end if
        end do
      end do
c
      return
      end if
c
c.... 2-D Problems
c
      if(naux.eq.3) then
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        yyj = yi(j)
        do i=1,npo
          xxi = xo(i)
          yyi = yo(i)
          r2 =  (xxi - xxj)*(xxi - xxj) +
     1          (yyi - yyj)*(yyi - yyj)
          if(r2.lt.eps) then
            B(i,j) = 0.0
          else
            B(i,j) = r2 * DLOG(r2) / 2.0
          end if
        end do
      end do
c
```

```
      return
      end if
c
c.... 3-D Problems
c
      if(naux.eq.4) then
c
c.... Evaluate the matrix of coefficients [B]
c
      do j=1,npi
        xxj = xi(j)
        yyj = yi(j)
        zzj = zi(j)
        do i=1,npo
          xxi = xo(i)
          yyi = yo(i)
          zzi = zo(i)
          r2 =  (xxi - xxj)*(xxi - xxj) +
     1          (yyi - yyj)*(yyi - yyj) +
     2          (zzi - zzj)*(zzi - zzj)
          if(r2.lt.eps) then
            B(i,j) = 0.0
          else
            B(i,j) = r2 * DLOG(r2) / 2.0
          end if
        end do
      end do
c
      return
      end if
c-----------------------------------------------------------------------
c     End of TPS
c-----------------------------------------------------------------------
c
      if(naux.ne.1.and.naux.ne.2.and.naux.ne.3.and.naux.ne.4) then
        write(*,*)'naux not properly defined in subroutine "bmatrx"'
        write(*,*)'naux =',naux
        STOP 'naux not equal to 1, 2, 3 or 4 in bmatrx'
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                        SUBROUTINE RMATRX
c
c     Generates the constraint matrix based on the basis functions
c
c     Created:        SEP26,95
c     Last Modified:
c
```

```
c***********************************************************************
c
      subroutine Rmatrx (no,naux,ni,npi,xi,yi,zi,
     1                   R)
c
      implicit         none
c
      integer          ni,no,npi,naux
      double precision xi(ni),yi(ni),zi(ni)
      double precision R(no,naux)
c
      integer          i
c
c.... Evaluate the matrix of constraints [R]
c
      do i=1,npi
        R(i,1) = 1.0
      end do
c
      if(naux.eq.2) then
        do i=1,npi
          R(i,2) = xi(i)
        end do
      end if
c
      if(naux.eq.3) then
        do i=1,npi
          R(i,2) = xi(i)
          R(i,3) = yi(i)
        end do
      end if
c
      if(naux.eq.4) then
        do i=1,npi
          R(i,2) = xi(i)
          R(i,3) = yi(i)
          R(i,4) = zi(i)
        end do
      end if
c
c.... END OF SUBROUTINE
c
      return
      end
c
c***********************************************************************
c
c                      SUBROUTINE FOTSXA
c
c     Transfer output data from the simple array used in MQ calculations
c     to the grid format of the driver (sxs,sys,szs)
c
c     Created:        SEP04,95
c     Last Modified:
```

```
c
c*************************************************************************
c
      subroutine fotsxa (npo,fo,ia,ja,ka,mode,sca)
c
      implicit      none
      integer       ia,ja,ka,mode
      integer       npo
      real          fo(npo)
      real          sca(ia,ja,ka,mode)
c
      integer       i,j,k,m
c
      m=1
      do k=1,ka
       do j=1,ja
        do i=1,ia
          sca(i,j,k,mode)=fo(m)
          m=m+1
        enddo
       enddo
      enddo
c
c
c.... END OF SUBROUTINE
c
      return
      end
c
c*************************************************************************
c
c                      SUBROUTINE MIMAX
c
c     Finds the minimum and the maximum among the elements of a vector
c
c     Created:        OCT24,95
c     Last Modified:
c
c*************************************************************************
c
      subroutine mimax (npi,xsi,ysi,zsi,ni,
     1                  xmin,xmax,ymin,ymax,zmin,zmax)
c
      implicit          none
      integer           ni,npi
      double precision xsi(ni),ysi(ni),zsi(ni)
      double precision xmin,xmax,ymin,ymax,zmin,zmax
c
      integer           i
      double precision auxx,auxy,auxz
c
c...... Find the min. and max. coordinate values along each direction
c        considering all the point involved in the sub-region (both input
c        as well as output points)
```

```
c
      do i=1,npi
        auxx = xsi(i)
        auxy = ysi(i)
        auxz = zsi(i)
        if(auxx.gt.xmax) xmax = auxx
        if(auxx.lt.xmin) xmin = auxx
        if(auxy.gt.ymax) ymax = auxy
        if(auxy.lt.ymin) ymin = auxy
        if(auxz.gt.zmax) zmax = auxz
        if(auxz.lt.zmin) zmin = auxz
      end do
c
c.... END OF SUBROUTINE
c
      return
      end
c
c---------------------------------------------
c This subroutine performs Lower - Upper (LU) decomposition
c
c Obtained From:
c   "Numerical Recipes", By  Press & Flannery et al.,
c      Cambridge University Press, 1989, p. 35-36
c a = matrix logical dimensions m by n
c     physical dimensions mp by np
c xx - row permutation effected by partial pivoting
c vv - internal
c-------------------------------------------------------------------
      SUBROUTINE LUDCMPd(A,N,NP,INDX,D,VV)
      implicit double precision (a-h,o-z)
      PARAMETER (NMAX=100,TINY=1.0E-20)
c      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      DIMENSION A(NP,NP),INDX(N),VV(NP)
      D=1.
      DO 12 I=1,N
        AAMAX=0.
        DO 11 J=1,N
          IF (dABS(A(I,J)).GT.AAMAX) AAMAX=dABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        DO 14 I=1,J-1
          SUM=A(I,J)
          DO 13 K=1,I-1
            SUM=SUM-A(I,K)*A(K,J)
13        CONTINUE
          A(I,J)=SUM
14      CONTINUE
        AAMAX=0.
        DO 16 I=J,N
          SUM=A(I,J)
```

```
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
15          CONTINUE
            A(I,J)=SUM
            DUM=VV(I)*dABS(SUM)
            IF (DUM.GE.AAMAX) THEN
              IMAX=I
              AAMAX=DUM
            ENDIF
16        CONTINUE
          IF (J.NE.IMAX)THEN
            DO 17 K=1,N
              DUM=A(IMAX,K)
              A(IMAX,K)=A(J,K)
              A(J,K)=DUM
17          CONTINUE
            D=-D
            VV(IMAX)=VV(J)
          ENDIF
          INDX(J)=IMAX
          IF(A(J,J).EQ.0.)A(J,J)=TINY
          IF(J.NE.N)THEN
            DUM=1./A(J,J)
            DO 18 I=J+1,N
              A(I,J)=A(I,J)*DUM
18          CONTINUE
          ENDIF
19      CONTINUE
        RETURN
        END
c
c-------------------------------------
c This subroutine performs Lower - Upper (LU) backsubstitution
c
c Obtained From:
c   "Numerical Recipes", By  Press & Flannery et al.,
c      Cambridge University Press, 1989, p. 37
c a = matrix logical dimensions m by n
c     physical dimensions mp by np
c xx - row permutation effected by partial pivoting
c b - on input contains contain RHS of A x = b
c     on output contains the solution x
c-------------------------------------------------------------------
        SUBROUTINE LUBKSBd(A,N,NP,INDX,B)
        implicit double precision (a-h,o-z)
        DIMENSION A(NP,NP),INDX(N),B(N)
        II=0
        DO 12 I=1,N
          LL=INDX(I)
          SUM=B(LL)
          B(LL)=B(I)
          IF (II.NE.0)THEN
            DO 11 J=II,I-1
              SUM=SUM-A(I,J)*B(J)
```

```
11          CONTINUE
          ELSE IF (SUM.NE.0.) THEN
            II=I
          ENDIF
          B(I)=SUM
12      CONTINUE
        DO 14 I=N,1,-1
          SUM=B(I)
          DO 13 J=I+1,N
            SUM=SUM-A(I,J)*B(J)
13        CONTINUE
          B(I)=SUM/A(I,I)
14      CONTINUE
        RETURN
        END

C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                     *
C*                   MULTIPLICATION OF TWO MATRICES                    *
C*                                                                     *
C*            Last Update: Nov 04, 92                                  *
C*                                                                     *
C*   OBJECTIVE :                                                       *
C*     Multiplication of A.B = C                                       *
C*                                                                     *
C*   INPUT :                                                           *
C*     A       : matrix n vs. m                                        *
C*     B       : matrix m vs. l                                        *
C*                                                                     *
C*   OUTPUT :                                                          *
C*     C       : matrix n vs. l, result of A.B                         *
C*                                                                     *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
        subroutine lsmpd (a,na,n,m,b,nb,l,c,nc)
c
        implicit          none
        integer           n,m,l,na,nb,nc
        double precision  a(na,m),b(nb,l)
        double precision  c(nc,l)
c
        integer           i,j,k
        double precision  aux
c
        do 200 i = 1,n
        do 200 j = 1,l
       aux = 0.0d0
       do 100 k = 1,m
          aux = aux + a(i,k) * b(k,j)
 100      continue
       c(i,j) = aux
 200    continue
c
        return
```

```
      end
C
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                        *
C*                    MULTIPLICATION OF TWO MATRICES                      *
C*                       [A] in band storage mode                         *
C*                                                                        *
C*                                                                        *
C*          Last Update: Dec 16, 92                                       *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                        *
C*  OBJECTIVE :                                                           *
C*    Multiplication of A.B = C                                           *
C*                                                                        *
C*  INPUT :                                                               *
C*    A      : matrix n vs. n (BAND STORAGE MODE)                         *
C*    B      : matrix n vs. l                                             *
C*                                                                        *
C*  OUTPUT :                                                              *
C*    C      : matrix n vs. l, result of A.B                              *
C*                                                                        *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      subroutine lsmpbd (a,n,band,b,l,c)
c
      implicit          none
c
      integer           n,band,l
      double precision  a(n,band), b(n,l)
      double precision  c(n,l)
c
      integer           i,j,k
      double precision  aux,baux
c
      do 100 j=1,l
      do 100 i=1,n
         c(i,j) = 0.0d0
 100  continue
c
      do 300 k=1,l
      do 300 i=1,n
     aux = 0.0d0
         do 200 j=1,band
        aux = aux + b(i+j-1,k)*a(i,j)
 200     continue
     c(i,k) = aux
 300  continue
c
      do 400 k=1,l
        do 400 i=1,n-band+1
          baux = b(i,k)
          do 400 j=2,band
        c(i-1+j,k) = c(i-1+j,k) + baux*a(i,j)
 400  continue
```

```
c
      do 500 k=1,l
      do 500 i=n-band+2,n-1
      baux = b(i,k)
      do 500 j=2,n-i+1
        c(i-1+j,k) = c(i-1+j,k) + baux*a(i,j)
 500  continue
c
      return
      end
c
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                   *
C*                  MULTIPLICATION OF TWO MATRICES                   *
C*                                                                   *
C*           Last Update: Mar 09, 93                                 *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C*                                                                   *
C*  OBJECTIVE :                                                      *
C*    Multiplication of A^T.B = C                                    *
C*                                                                   *
C*  INPUT :                                                          *
C*    A       : matrix n vs. m                                       *
C*    B       : matrix n vs. l                                       *
C*                                                                   *
C*  OUTPUT :                                                         *
C*    C       : matrix m vs. l, result of A.B                        *
C*                                                                   *
C* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
      subroutine lsmptd (a,na,n,m,b,nb,l,c,nc)
c
      implicit         none
      integer          n,m,l,na,nb,nc
      double precision a(na,m),b(nb,l)
      double precision c(nc,l)
c
      integer          i,j,k
      double precision aux
c
      do 200 i = 1,m
      do 200 j = 1,l
     aux = 0.0d0
     do 100 k = 1,n
        aux = aux + a(k,i) * b(k,j)
 100    continue
     c(i,j) = aux
 200  continue
c
      return
      end
      subroutine nurbs(mode,f)

      implicit double precision (a-h,o-z)
```

```
      include 'csdcfd.par'
      parameter(ndom=2,ndim=6,ndep=6,nwork=NWK)
      parameter(maxc=(ndim+1)*imxs*jmxs)
c
c      xs,ys,zs    - structural grid points
c      sxs,sys,szs - structural mode shapes values at grid points
c      xa,ya,za    - aerodynamic grid points
c      sxa,sya,sza - calculated aerodynamic deflections
c      isg,jsg,ksg - number of structural points
c      is,js,ks    - number of structural points ( for each mode)
c      ia,ja,ka    - number of aerodynamic points
c
c
        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg
       common /aero/ xa(dnaro),ya(dnaro),za(dnaro),fa(dnaro),npts
       common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       common /cnt/ ia,ja,ka
      common /para/ npt_s,npt_t,sdir(imxs*jmxs),tdir(jmxs)
c
c Local Variables
c     uo - array for aerodynamic grid points and mode shapes
c     u  - array for structural grid points and mode shapes
c     f  - array for structural grid points to be parameterized
c     st - parameterized values for the structural grid
c     sttest - parameterized value of the aerodynamic grid
c     work   - scratch array for dtnurbs subroutines
c     NWK    - dimension of scratch array
c     v      - value of spline at wanted location
c     out1,2,3 -  holding arrays for spline output values
c     ipdir - holding array for the plane direction
c
c
      double precision uo(imxa ,jmxa ,ndim), v(10)
      double precision work(NWK), u(imxs,jmxs,ndim),snpvl(2),
     &pt(3),p(2),s_carray(maxc)
       dimension f(dnaro,3)
c          for fitsurf
      dimension c_tmp((ndim+1)*jmxs*imxs)
      integer npts_curv(jmxs)
c
      integer ipdir(dnaro)
      dimension xd(4),yd(4),zd(4),fd(4),tmp(dnstr),tmp2(dnstr)

c-----------------------------------------------------------------------
c Logicals
c-----------------------------------------------------------------------
c
c  npi   - number of point in CSD grid
c  ndim  - number of dependant variables
c  ndeg  - degree of spline
c
      ipar=2
```

```
c       write(*,*) ' Enter the spline degree '
c       read(*,*) ndeg
c      write(*,*)' Enter the spline degree in the s direction'
c      read(*,*)ndeg(1)
c      write(*,*)' Enter the spline degree in the t direction'
c      read(*,*)ndeg(2)
c      ndeg(1)=3
c      ndeg(2)=3
       ndeg=3
c
c  Rearrange structural grid and mode information
c
      m=0
      do k=1,ksg
        do j=1,jsg
          do i=1,isg
            m=m+1
            u(i,j,1)=xt(m)
            u(i,j,2)=yt(m)
            u(i,j,3)=zt(m)
            u(i,j,4)=sxs(i,j,k)
            u(i,j,5)=sys(i,j,k)
c            u(i,j,4)=0.0
c            u(i,j,5)=0.0
            u(i,j,6)=szs(i,j,k)
          enddo
        enddo
      enddo
c
c Rearrange aerodynamic grid information
c
      m=0
      do k=1,ka
        do j=1,ja
          do i=1,ia
            m=m+1
            uo(i,j,1)=xa(m)
            uo(i,j,2)=ya(m)
            uo(i,j,3)=za(m)
          enddo
        enddo
      enddo
c
c     Use surface fitting routine for non-regular CSD grid
c
c ========================================================================
c
c...this program will fit a NURB surface to offset points.
c
c  Version:     6.0   Using DT_NURBS for fitting
c  Date:        2/1991
c  Programmer:  Bob Ames, DTRC
c  Modified for use in FASIT 8/95 & 9/95 by MJ Smith
```

```
c
c  Process...
c  Offset data is read in for each curve.  Each curve may have any number
c
c MEMORY:
c This program can COMSUME memory if you're not careful!  See fitsurf.inc.
c
c
c ==========================================================================


c ***
c Start loop for each curve
c ***
         do  j=1,jsg

c Get number of points...check limits
c
           npts_curv(j)=isg
c
           if (npts_curv(j) .gt. imxs) then
               write(6,*) ' Error max mumber of points per '
               write(6,*) ' curve exceeded!  Terminating..!'
               stop
           end if

c Find curve with maximum number of points
           if(j .eq. 1) then
              maxpts= npts_curv(j)
           else
              maxpts=max0(npts_curv(j), maxpts)
           end if
c
           do  k=1,npts_curv(j)
c Check for coincident points
             if (k .gt. 1) then
c Assume points coincident unless proven wrong
                icoin=0
                do i=1,ndep
                   if (u(k,j,i).eq.u(k-1,j,i)) then
                       icoin=icoin+1
                   else
                       continue
                   end if
                end do
                if (icoin.eq.ndep) then
                  write(6,*) 'Error... input file'
                  write(6,*) 'contains coincident points'
                  write(6,*) ' '
c                 write(6,fmt='('Curve=',i3,' Point=',i3)')
c    &                                   j,k
                  stop
                end if
```

```
            end if
          end do         !end npts
        end do           !end ncurv
c
c           Fit surface to input data
c
      call surface(jsg,npts_curv,ndeg,maxpts,u,s_carray)
       do i=1,100
         write(6,*) ' s_carray ',i,s_carray(i)
       enddo
c
c           end of original fitsurf program
c
c*******************************************************************
c           Manipulate the Unknown Function Grid to Determine
c                the Parameterization
c*******************************************************************
        m=0
        do i=1,ia-1
        do j=1,ja-1
           m=m+1
           jp1=i*ia+j
           xd(1)=xa(m)
           yd(1)=ya(m)
           zd(1)=za(m)
           xd(2)=xa(m+1)
           yd(2)=ya(m+1)
           zd(2)=za(m+1)
           xd(3)=xa(jp1)
           yd(3)=ya(jp1)
           zd(3)=za(jp1)
           xd(4)=xa(jp1+1)
           yd(4)=ya(jp1+1)
           zd(4)=za(jp1+1)
           call plane1(xd,yd,zd,4,ipdir(m),iplane)
           write(6,*) ' plane', m,ipdir(m),iplane
        enddo
        enddo
        do m=1,npts
c
c           Now search for the known panel which encloses
c           the point, based on the direction of the data panel
c
        call search(ipdir(m),lout,mout,iout,jout,
     1       xa(m),ya(m),za(m),xd,yd,zd)
        write(6,*) ' search ',m,lout,mout,iout,jout
c
c           Now call the bivariate interpolation or extrapolation
c               routines
c
c                  interpolation
         incx=1
         if(lout.gt.0) then
           fd(1)=sdir(mout)
```

```
            fd(2)=sdir(mout+1)
            fd(4)=sdir(mout+isg)
            fd(3)=sdir(mout+isg+1)
            xx=xa(m)
            yy=ya(m)
            do ll=1,4
              write(6,*) ' 1 fd ',xd(ll),yd(ll),fd(ll)
            enddo
            call bivarin(xd,yd,fd,xx,yy,z,ierr)
            write(6,*) ' xx ',xx,yy,z
            snpvl(1)=z
            fd(1)=tdir(jout)
            fd(2)=tdir(jout)
            fd(4)=tdir(jout+1)
            fd(3)=tdir(jout+1)
            xx=xa(m)
            yy=ya(m)
            call bivarin(xd,yd,fd,xx,yy,z,ierr)
            snpvl(2)=z
            write(6,*) ' calling dtnpvl ',snpvl(1),snpvl(2)
            call dtnpvl(snpvl,incx,s_carray,work,NWK,v,ier)
            do i=1,4
             write(6,*) ' int ',xd(i),yd(i),fd(i)
            enddo
            do i=1,6
              write(6,*) ' dtnpvl ',i,v(i)
            enddo
            f(m,1)=v(4)
            f(m,2)=v(5)
            f(m,3)=v(6)
           else if (lout.eq.-1) then
c                extrapolate along x direction
c            do l=1,3
              l=3
              mout=((jout-1)*isg)+iout
              mp1=(jout*isg)+iout
              do ll=1,isg
              tmp(ll)=szs(ll,jout,1)
              tmp2(ll)=xt((jout-1)*isg+ll)
              write(101,*) ' a ',ll,tmp(ll),tmp2(ll)
              enddo
              xx=xa(m)
              call polint(tmp2,tmp,isg,xx,fd(1),err)
              if(l.eq.3) then
              write(101,*) 'polint ',m, xa(m),fd(1)
              write(101,*) (xt(ll),ll=nout,nout+isg),
     1          (tmp(i),i=1,isg)
              endif
c
              do ll=1,isg
              tmp(ll)=szs(ll,jout+1,1)
              tmp2(ll)=xt(jout*isg+ll)
              write(101,*) ' b ',ll,tmp(ll),tmp2(ll)
              enddo
```

```
             xx=xa(m)
             call polint(tmp2,tmp,isg,xx,fd(4),err)
             xd(1)=xa(m)
             xd(2)=xt(mout)
             xd(3)=xt(mp1)
             xd(4)=xa(m)
             yd(1)=yt(mout)
             yd(2)=yt(mout)
             yd(3)=yt(mp1)
             yd(4)=yt(mp1)
             fd(2)=u(iout,jout,l+3)
             fd(3)=u(iout,jout+1,l+3)
             xx=xa(m)
             yy=ya(m)
             call bivarin(xd,yd,fd,xx,yy,z,ierr)
             f(m,l)=z
c            enddo
          else if (lout.eq.-2) then
c                 extrapolate along y direction
c           do l=1,3
             l=3
             mout=((jout-1)*isg)+iout
             do ll=1,jsg
             tmp(ll)=szs(iout,ll,1)
             tmp2(ll)=yt((ll-1)*isg+iout)
             enddo
             yy=ya(m)
             call polint(tmp2,tmp,jsg,yy,fd(1),err)
             if(l.eq.3) then
c             write(6,*) 'polint ',m, xa(m),fd(1)
c             write(6,*) (xt(ll),ll=nout,nout+isg),
c     1          (tmp(i),i=1,isg)
             endif
c
             do ll=1,jsg
             tmp(ll)=szs(iout+1,ll,1)
             tmp2(ll)=yt((ll-1)*isg+iout+1)
             enddo
             yy=ya(m)
             call polint(tmp2,tmp,jsg,yy,fd(4),err)
             mout=((jout-1)*isg+iout)
             xd(1)=xt(mout)
             xd(2)=xt(mout)
             xd(3)=xt(mout+1)
             xd(4)=xt(mout+1)
             yd(1)=ya(m)
             yd(2)=yt(mout)
             yd(3)=yt(mout+1)
             yd(4)=ya(m)
             fd(2)=u(iout,jout,l+3)
             fd(3)=u(iout+1,jout,l+3)
             xx=xa(m)
             yy=ya(m)
             call bivarin(xd,yd,fd,xx,yy,z,ierr)
```

```
            f(m,l)=z
c           enddo
         else if (lout.eq.-3) then
c                extrapolate in a corner
c          do l=1,3
           l=3
           mout=((jout-1)*isg)+iout
           mp1=(jout*isg)+iout
           do ll=1,isg
           tmp(ll)=szs(ll,jout,1)
           tmp2(ll)=xt((jout-1)*isg+ll)
           enddo
           xx=xa(m)
           call polint(tmp2,tmp,isg,xx,fd(1),err)
           if(l.eq.3) then
c           write(6,*) 'polint ',m, xa(m),fd(1)
c           write(6,*) (xt(ll),ll=nout,nout+isg),
c     1        (tmp(i),i=1,isg)
           endif
c
           do ll=1,jsg
           tmp(ll)=szs(iout,ll,1)
           tmp2(ll)=yt((ll-1)*isg+iout)
           enddo
           yy=ya(m)
           call polint(tmp2,tmp,jsg,yy,fd(4),err)
           f(m,l)=((fd(1)-szs(iout,jout,1))+
     1       (fd(4)-szs(iout,jout,1)))+szs(iout,jout,1)
           write(102,*) ' fd ',m,l,f(m,l)
c           end do m=1,npts
        endif
        enddo
c
        do i=1,dnaro
         write(105,*) i,f(i,3)
        enddo
         return
         end


      subroutine surface (ncurv,npts_curv,ndeg,maxpts,xyz,s_carray)
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cc
c
c  Date:        7/1990
c  Programmer:  Bob Ames, DTRC
c
c
c  --------------------
c  Variable definitions
c  --------------------
c
c  carray     - array holding nurb data
c  c_tmp      - temp holding array of individual curves
```

```
c  hold        - scratch space for DT_NURBS
c  maxpts      - maximum number of points for any input curve(i)
c  ndeg        - degree of curve or surface (order=ndeg+1)
c  ndep        - number of dependant variables allowed (x,y,z,cp,vx,vy,vz...)
c  ndom        - number of independant variables (s,t...)
c  npt_gpar    - number of data points for each ndom
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c Include parameters
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(ndom=2,ndim=6,ndep=6,nwork=NWK)
c DTGPAR variables
      common /para/ npts,nptt,sdir(imxs*jmxs),tdir(jmxs)
      parameter(maxc=(ndim+1)*imxs*jmxs)
      dimension xyz(imxs,jmxs,ndim)
      dimension temp(imxs*ndep),c_tmp(maxc),s_carray(maxc)
      dimension t_gpar(imxs*ndom*ndep),s_gpar(jmxs*ndom*ndep)
      dimension c_carray(maxc),hold(NWK),work(NWK)
      integer npt_gpar(ndom),ipntr_s(jmxs*ndep*2),len_c
      integer  npts_curv(jmxs),icc

      logical diag

c ***
c Set initial values
c ***

c Other logicals

       diag= .false.
       diag= .true.
c *******************************************************************
c Generate parametric array of initial input data in preparation for
c curve fitting
c *******************************************************************
c ***
c Load temp array with input data from data base
c ***
      ipntr_s(1)=1
      ndep_gpar=3
      npts=0
      nptt=ncurv
      do j=1,ncurv
         index=1
         do m=1,ndep_gpar
            do k=1,npts_curv(j)
               temp(index)=xyz(k,j,m)
               index=index+1
            end do
         end do

c Diagnostics on temp array..(i.e. input data)
```

```
        if (diag .eq. .true.) then
           write(6,*) ' '
           write(6,*) '========== NEW CURVE ========== '
           write(6,*) 'Data for curve number=',j
           write(6,*)  '    k   m  indx   input data'
           index=1
           do m=1,ndep_gpar
              do k=1,npts_curv(j)
                write(6,fmt='(1x,3i4,f12.6)') k,m,index,
     &                                 temp(index)
                index=index+1
              end do
           end do
        end if
c End Diagnostics

c ***
c Set up dtgpar arguments
c ***
        npt_gpar(1) = npts_curv(j) !Number of data points in s direction
        ndom_gpar   = 1            !Number of indep variables (i.e. s,t)
c        ndep_gpar   = 1            !Number of depend variables (i.e. x,y,z)
        ndim_gpar   = npts_curv(j)  !Max parameter dimension
c
        call dtgpar (npt_gpar,temp,ndom_gpar,ndep_gpar,work,nwork,
     &             ndim_gpar,t_gpar,ier)
        do l=1,npt_gpar(1)
           ll=npts+l
           sdir(ll)=t_gpar(l)
        enddo
        npts=npt_gpar(1)+npts
        if (ier .ne. 0) then
           write(6,*) ' DTGPAR returned IER=',ier
           stop
        end if

c Diagnostics
        if (diag .eq. .true.) then
           write(6,*) ' '
           write(6,*) '*** DIAG. ON T_GPAR ***'
c          write(6,fmt='(1x,'(1) Curve number=',i3)') j
           do loop2=1,ndim_gpar
              write(6,fmt='(1x,i6,2f12.6)')
     &        loop2,t_gpar(loop2)
           end do
        end if
c End Diagnos

c************************************************************************
c Fit input curve pts using DTNSI
c************************************************************************
        do ii=1,ndep
           if (ii. eq. 1) then
              icc=-1
```

```
          else
             icc=2
          end if

c Load input array of dependant variables
          do kk=1,npts_curv(j)
             temp(kk)=xyz(kk,j,ii)
             write(6,*) ' dtnsi ',kk,ii,temp(kk)
          end do
          k=ndeg+1
          ncoef=(npts_curv(j)-2)+k
          nc=5+(ndep+1)*ncoef+k
          call dtnsi(npts_curv(j),
     &                 t_gpar,
     &                 temp,ndeg,icc,hold,nwork,maxc,
     &                 c_carray,nc,ier)
       end do
       do ll=1,100
         write(6,*) ' c_array ',ll,c_carray(ll)
       enddo

c Diagnostics
       if (diag .eq. .true.) then
          call fit_diag(c_carray,j)
       end if
c Error Check
       if (ier .lt. 0) then
           write(6,*) ' DTNSI returned IER=',ier
           stop
       end if

c*************************************************************************
c Generate points at constant parametric locations using max number of
c points contained in any one input curve...load back into data base
c*************************************************************************
       call dtsepp(c_carray,maxpts,work,nwork,temp,ier)

c ***
c Reload surf data base with new refitted data
c ***
       npts_curv(j)=maxpts

       incr=1
       do ii=1,ndep
          do kk=1, maxpts
            xyz(kk,j,ii)=temp(incr)
             write(6,*) ' new temp ',ii,kk,incr,xyz(kk,j,ii)
            incr=incr+1
          end do
       end do

c*************************************************************************
c Generate parametric array based on the same knot set for each curve
c*************************************************************************
```

```
        do i=1,maxpts
          t_gpar(i)=
     &               real((i-1)/(maxpts-1.0))
        end do

c Diagnostics
        if (diag .eq. .true.) then
           write(6,*) ' '
           write(6,*) '*** DIAG. ON T_GPAR ***'
c          write(6,fmt='(1x,' Curve number=',i3)') j
           do loop2=1,maxpts
              write(6,fmt='(1x,i6,f12.6)')
     &        loop2,t_gpar(loop2)
           end do
        end if
c End Diagnostics

c*********************************************************************
c Fit new interpolated input points using DTNSI
c*********************************************************************
        do ii=1,ndep
           if (ii. eq. 1) then
              icc=-1
           else
              icc=2
           end if

c Load input array of dependant variables into temp array
           do kk=1,npts_curv(j)
              temp(kk)=xyz(kk,j,ii)
           end do

           call dtnsi(npts_curv(j),
     &                t_gpar,
     &                temp,ndeg,icc,hold,nwork,maxc,
     &                c_carray,
     &                len_c,ier)

c Error Check
           if (ier .lt. 0) then
              write(6,*) ' DTNSI returned IER=',ier
              stop
           end if

c Check size of C array against what is output
           call dtcsiz(c_carray,isize,ier)
           if (ier .ne. 0) then
              write(6,*) ' DTCSIZ returned IER=',ier
           end if
           if (isize .ne. len_c) then
              write(6,*) 'C size does not match!!'
              stop
           end if
```

```
         end do

c Diagnostics
         if (diag .eq. .true.) then
            call fit_diag(c_carray,j)
         end if
c End Diagnostics


c ***
c Load curve data into tmp array
c ***

         do nloop=1,len_c
            c_tmp(ipntr_s(j) + nloop-1) =
     &      c_carray(nloop)
            ich=ipntr_s(j)+nloop-1
         end do

c Set pointer into tmp for each location of next curve
         if (j .ne. ncurv) then
            ipntr_s(j+1) = len_c + ipntr_s(j)
         end if
c***
      end do        !end curve loop ...end of surface(i)


c ***
c Store first point of each section to parameterize t space
c ***


c Use first point of station...pnt(1)
      index=1
      do m=1,ndep_gpar
         do i=1,ncurv
            temp(index)=xyz(1,i,m)
            write(6,*) ' t temp ',i,m,temp(index)
            index=index+1
         end do
      end do

c Diagnostics on temp array..(i.e. input data)
      if (diag .eq. .true.) then
         write(6,*) ' '
         write(6,*) ' '
         write(6,*) '========== T CURVE ========== '
c        write(6,fmt='(1x,'Data for T parameter =')')
         write(6,*) '     k    m    indx    input data'
         index=1
         do m=1,ndep_gpar
            do k=1,ncurv
              write(6,fmt='(1x,3i4,f12.6)') k,m,index,
```

```
     &                           temp(index)
              index=index+1
           end do
        end do
     end if
c End Diagnostics


c Set up dtgpar arguments
      npt_gpar(1) = ncurv          !Number of data points in t direction
      ndom_gpar   = 1                       !Number of indep variables (i.e.
s,t)
c      ndep_gpar   = 1              !Number of depend variables (i.e. x,y,z)
      ndim_gpar   = ncurv          !Max parameter dimension

      call dtgpar (npt_gpar,temp,ndom_gpar,ndep_gpar,work,nwork,
     &       ndim_gpar,s_gpar,ier)
      do l=1,npt_gpar(1)
         tdir(l)=s_gpar(l)
         write(6,*) ' tdir ',l,tdir(l)
      enddo
      if (ier .ne. 0) then
         write(6,*) ' DTGPAR returned IER=',ier
         stop
      end if

c Diagnostics
      if (diag .eq. .true.) then
         write(6,*) ' '
         write(6,*) '*** DIAG. ON NEW T_GPAR ***'
c        write(6,fmt='(1x,'(1) Curve number=',i3)') j
         do loop2=1,ndim_gpar
            write(6,fmt='(1x,i6,2f12.6)')
     &      loop2,s_gpar(loop2)
         end do

c Check on new c_tmp array
         write(6,*) ' '
         write(6,*) ' '
         write(6,*) ' **** C_TMP **** '
         call fit_diag(c_tmp,999)

      end if



c ***********************************************************************
c Fit surface to curves
c ***********************************************************************

      call dtcrbl(c_tmp,len_c,
     &            s_gpar,
     &            ncurv, ndeg, work, nwork,
     &            s_carray, ier)
```

```
      do i=1,100
        write(6,*) ' s_carray ',i,s_carray(i)
      enddo
      if (ier .ne. 0) then
         write(6,*) ' DTCRBL returned IER=',ier
         if ((ier .eq. 1) .or. (ier .eq. 2)) then
            write(6,*) ' ier=1 => some illconditioning'
            write(6,*) ' ier=2 => severe illconditioning'
         else
            stop
         endif
      end if

c Diagnostics
      if (diag .eq. .true.) then
         call fit_diag(s_carray,id)
      end if

c Determine length of C array
      call dtcsiz(s_carray,len_c,ier)
      if (ier .ne. 0) then
         write(6,*) ' DTCSIZ returned IER=',ier
      end if

c
      return
      end
      subroutine fit_diag(carray,id)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(ndim=6,ndep=6)
      parameter(maxc=(ndim+1)*imxs*jmxs)
      dimension carray(maxc)

      double precision plo(2),phi(2)
      integer ploc(3),qloc(4)
      integer kord(2),ncoef(2)

      call dtget(carray,.true.,2,n,mraw,mdep,kord,ncoef,plo,phi,ier)

      write(6,*) ' '
      write(6,*) ' *** DTGET ON C ARRAY ***'
      write(6,fmt='(1x,/,
     &      ''(1) Curve number=                              '',i3,/,
     &      ''(2) No. indep. variables=                      '',i3,/,
     &      ''(3) No. dep. variables c(2)=                   '',i3,/,
     &      ''(4) No. dep. variables {c(2)-1 if c(2) neg.}='',i3,/,
     &      ''(5) Order of spline U and V=                   '',2i5,/,
     &      ''(6) No. of basis funtions in U and V=          '',2i10,/,
     &      ''(7) IER error flag=                            '',i10,/,
     &      ''(8) Parameter limit low=                       '',2f12.6,/,
     &      ''(9) Parameter limit high=                      '',2f12.6)')
     &        id,n,mraw,mdep,kord,ncoef,ier,plo,phi
      return
```

```
      end
c      ssssssssssssssssssssssssssssssssssss
      subroutine load(mode,idir,nspts,ft)
c      ssssssssssssssssssssssssssssssssssss

      implicit double precision (a-h,o-z)
      include 'csdcfd.par'

        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg

      dimension ft(dnstr+3)


c
        m=0
        do k=1,ksg
          do j=1,jsg
            do i=1,isg
              m=m+1
              if(idir.eq.1) ft(m)=sxs(i,j,k)
              if(idir.eq.2) ft(m)=sys(i,j,k)
              if(idir.eq.3) ft(m)=szs(i,j,k)
            enddo
          enddo
        enddo
c
      return
      end
      subroutine search(ipdir,lout,mout,iout,jout,xa,ya,za,xs,ys,zs)

      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
c
c      xs,ys,zs     - structural grid points
c      sxs,sys,szs - structural mode shapes values at grid points
c      xa,ya,za     - aerodynamic grid points
c      sxa,sya,sza - calculated aerodynamic deflections
c      isg,jsg,ksg - number of structural points
c      is,js,ks     - number of structural points ( for each mode)
c      ia,ja,ka     - number of aerodynamic points
c
c
        common /struct/ sxs(imxs,jmxs,kmxs)
     &,sys(imxs,jmxs,kmxs),szs(imxs,jmxs,kmxs),is
     &,js,ks,isg,jsg,ksg
       common /str/ xt(dnstr),yt(dnstr),zt(dnstr),ft(dnstr+3),nspts
       dimension xs(4),ys(4),zs(4)

       iout=0
       jout=0
       lout=0
       mout=0
```

```
      go to (300,200,100) ipdir
c
c           x-y plane (ipdir=3)
c
  100 continue
      do l=1,jsg-1
      do i=1,isg-1
        m=(l-1)*isg+i
        mp1=m+isg
c        write(6,*) ' xy ',xt(m),xa,xt(m+1)
        if(xt(m).gt.xa) go to 150
        if(xt(m).le.xa.and.xt(m+1).ge.xa) then
            if(yt(m).gt.ya.and.yt(m+1).gt.ya) go to 120
            if(yt(m).le.ya.and.yt(mp1).ge.ya) then
                mout=m
                jout=l
                iout=i
                lout=1
                go to 500
            endif
            if(yt(m+1).le.ya.and.yt(mp1+1).ge.ya) then
                mout=m
                jout=l
                iout=i
                lout=1
                go to 500
            endif
        endif
  150   continue
      enddo
  120       continue
      enddo
c        compute the extrapolation locations
      lout=0
      mout=0
      m=1
      if(xt(m).gt.xa.or.xt(m+isg-1).lt.xa) then
       if(xt(m).gt.xa) iout=1
       if(xt(m+isg-1).lt.xa) iout=isg
       if(yt(iout).gt.ya) then
            jout=1
            lout=-3
             return
       endif
       do j=1,jsg-1
           m=(j-1)*isg+iout
           if(yt(m).le.ya.and.yt(m+isg).ge.ya) then
                   lout=-1
                   jout=j
                   return
           endif
        enddo
       if (yt((jsg-1)*isg+iout).lt.ya) then
              jout=jsg
```

```
              lout=-3
              return

       endif
      endif
c                    outside span
      m=1
      if(yt(m).gt.ya.or.yt((jsg-1)*isg).lt.ya) then
       if(yt(m).gt.ya) jout=1
       if(yt((jsg-1)*isg+1).lt.ya) jout=jsg
       if(xt(m).gt.(xa)) then
            iout=1
            lout=-3
            return
       endif
       if(xt((jsg-1)*isg+isg).lt.xa) then
            iout=isg
            lout=-3
            return
       endif
       do i=1,isg-1
           m=(jout-1)*isg+i
           if(xt(m).le.xa.and.xt(m+1).ge.xa) then
                   lout=-2
                   iout=i
                   return
           endif
       enddo
      endif
      write(6,*) 'error in search', xa, ya
      return
c           y-z plane (ipdir=2)
 200  continue
      do l=1,jsg-1
      do i=1,isg-1
        m=(l-1)*isg+i
        mp1=m+isg
        if(zt(m).gt.za) go to 250
        if(zt(m).le.za.and.zt(m+1).ge.za) then
          iout=i
            if(yt(m).gt.ya) go to 220
            if(yt(m).le.ya.and.yt(mp1).ge.ya) then
                jout=l
                mout=m
                lout=1
                go to 500
            endif
 220        continue
        endif
 250    continue
      enddo
      enddo
      write(6,*) 'error ',ya,za
      lout=0
```

```
      return
c           x-z plane (ipdir=1)
  300 continue
      do l=1,jsg-1
      do i=1,isg-1
        m=(l-1)*isg+i
        mp1=m+isg
        if(xt(m).gt.xa) go to 350
        if(xt(m).le.xa.and.xt(m+1).ge.xa) then
            if(zt(m).gt.za) go to 320
            if(zt(m).le.za.and.zt(mp1).ge.za) then
                jout=l
                iout=i
                mout=m
                lout=1
                go to 500
            endif
  320       continue
        endif
  350   continue
      enddo
      enddo
      write(6,*) 'error ',xa,za
      lout=0
      return
c           load structural panel
  500 continue
      jp=mout+isg
      jp1=mout+isg+1
      xs(1)=xt(mout)
      ys(1)=yt(mout)
      zs(1)=zt(mout)
      xs(2)=xt(mout+1)
      ys(2)=yt(mout+1)
      zs(2)=zt(mout+1)
      xs(4)=xt(jp)
      ys(4)=yt(jp)
      zs(4)=zt(jp)
      xs(3)=xt(jp1)
      ys(3)=yt(jp1)
      zs(3)=zt(jp1)
c
c           final checks
c
      if(ipdir.eq.2) go to 600
      if(xa.ge.xs(4).and.xa.le.xs(3)) go to 600
      xr=(xs(4)-xs(1))*(ya-ys(1))/(ys(4)-ys(1))
      if(xr.le.xa) go to 600
      xr=(xs(3)-xs(2))*(ya-ys(2))/(ys(3)-ys(2))
      if(xr.ge.xa) go to 600
      if(iout.eq.1) then
           lout=0
           return
      endif
```

```
      mout=mout-1
      go to 500
600   continue
      if(ipdir.eq.1) go to 700
      if(ya.ge.ys(1).and.ya.ge.ys(2)) go to 700
      yr=(ys(2)-ys(1))*(xa-xs(1))/(xs(2)-xs(1))
      if(yr.le.ya) go to 700
      yr=(ys(3)-ys(4))*(xa-xs(4))/(xs(3)-xs(4))
      if(yr.ge.ya) go to 700
      if(jout.eq.1) then
           lout=0
           return
       endif
      mout=mout-isg
      go to 500
700   continue
      if(ipdir.eq.3) go to 800
      if(za.ge.zs(4).and.za.le.zs(3)) go to 800
      zr=(zs(4)-zs(1))*(xa-xs(1))/(xs(4)-xs(1))
      if(zr.le.za) go to 800
      zr=(zs(3)-zs(2))*(xa-xs(2))/(xs(3)-xs(2))
      if(zr.ge.za) go to 800
      if(iout.eq.1) then
           lout=0
           return
       endif
      mout=mout-1
      go to 500

800   return
      end
c
      subroutine bivarin(xd,yd,fd,x,y,f,ierr)

c  Programmer: V. M. Kaladi, Nov 2, '93

      implicit double precision (a-h,o-z)

      parameter (eps=1e-5, itmax=500, eps2=1e-3)
      dimension xd(4),yd(4),fd(4)
      dimension a(2,2)

c Given the values of a function f(x,y) at four points
c ((xd(i),yd(i), i=1,4) as fd(i), i=1,4 and the point
c (x,y) inside the quadrilateral formed by the data points,
c this subroutine computes the linear bivariate interpolated value
c of f(x,y).

c statement functions
c g(u,v)= sum_(j=1,4) of gj*psij(u,v)
c       = (g0 + ga*u + gb*v + gc*u*v)
c where gj are values at the data points.
      g0(g1,g2,g3,g4)= 0.25*( g1 + g2 + g3 + g4)
      ga(g1,g2,g3,g4)= 0.25*(-g1 + g2 + g3 - g4)
```

```
      gb(g1,g2,g3,g4)= 0.25*(-g1 - g2 + g3 + g4)
      gc(g1,g2,g3,g4)= 0.25*( g1 - g2 + g3 - g4)
      gd(g1,g2,g3,g4)= ((g1-g2)*(g1-g2)+(g3-g4)*(g3-g4))
c end statement functions
c
c check to see if there are identical points
c
      do l=1,4
        chk = gd(xd(l),x,yd(l),y)
c         chk=(xd(l)-x)*(xd(l)-x)+(yd(l)-y)*(yd(l)-y)
        if(sqrt(chk).le.eps) then
          f=fd(l)
          return
        endif
      enddo

      ierr= 0

      x0= g0(xd(1),xd(2),xd(3),xd(4))
      y0= g0(yd(1),yd(2),yd(3),yd(4))
      xa= ga(xd(1),xd(2),xd(3),xd(4))
      ya= ga(yd(1),yd(2),yd(3),yd(4))
      xb= gb(xd(1),xd(2),xd(3),xd(4))
      yb= gb(yd(1),yd(2),yd(3),yd(4))
      xc= gc(xd(1),xd(2),xd(3),xd(4))
      yc= gc(yd(1),yd(2),yd(3),yd(4))

      up= 0.0
      vp= 0.0
      iter=0
 10   continue
        a(1,1)= xa + xc*vp
        a(1,2)= xb
        a(2,1)= ya
        a(2,2)= yb + yc*up
        if((a(1,1)*a(2,2) - a(2,1)*a(1,2)).eq.0.0) go to 20
        deti= 1.0/(a(1,1)*a(2,2) - a(2,1)*a(1,2))
        u= (a(2,2)*(x-x0) - a(1,2)*(y-y0))*deti
        v= (a(1,1)*(y-y0) - a(2,1)*(x-x0))*deti
        errnorm= dsqrt(  (u-up)**2 + (v-vp)**2 )
        up=u
        vp=v
        iter= iter+1
      if (errnorm .gt. eps .and. iter .lt. itmax) go to 10

      if (iter .ge. itmax) then
        print*,'Error, max no. of iterations exceeded '
        print*,'in subroutine bivarin'
        ierr= 1
        write(6,*) ' errnorm = ',errnorm,' eps = ',eps
        write(6,*) ' points ',xa,ya
        write(6,*) ' xd ',xd(1),xd(2),xd(3),xd(4)
        write(6,*) ' yd ',yd(1),yd(2),yd(3),yd(4)
        stop
```

```
      end if

c      if (u .lt. -1.0 .or. u .gt. 1. .or. v .lt. -1 .or. v .gt. 1) then
c        print*,'Error, interpolation point (x,y) is not within '
c        print*,'the quadrilateral of data points.'
c        err= .true.
c        return
c      end if

c Assuming any point falling outside the quadrilateral of data points
c is due to round off error the point is reset to the nearest boundary.
      if (abs(u).gt. 1.0) u= int(u)
      if (abs(v).gt. 1.0) v= int(v)

      f=     g0(fd(1),fd(2),fd(3),fd(4)) +
     &   u * ga(fd(1),fd(2),fd(3),fd(4)) +
     &   v * gb(fd(1),fd(2),fd(3),fd(4)) +
     &   u*v*gc(fd(1),fd(2),fd(3),fd(4))
c
c
      t = (x -xd(1))/(xd(2)-xd(1))
      u = (y -yd(1))/(yd(4)-yd(1))
      fa= (1.-t)*(1.-u)*fd(1) + t*(1.-u)*fd(2) + t*u*fd(3)
     1    + (1.-t)*u*fd(4)

      return
c           The det is 0., so we missed an identical point the
c               first time through
  20  continue

c check to see if there are identical points
c
      mchk=0
      chkmin=9.999e5
      do l=1,4
        chk = gd(xd(l),x,yd(l),y)
        if(chk.lt.chkmin) then
             chkmin=chk  '
             mchk=l
        endif
      enddo
      f=fd(mchk)
      return
      end
C -----------------------------------------------------------------------
      subroutine plane1(x,y,z,n,ipdir,planar)
      implicit double precision (a-h,o-z)
C This routine checks to see if the surface resides within a single plane
C It uses input structural coordinates
C planar = .true.
C        = .false.
C -----------------------------------------------------------------------
      integer ipdir, planar
      dimension x(n), y(n), z(n)
```

```
c
c local variables
c
      dimension a(3), b(3), c(3), d(3)
c
c external functions
c
      double precision fmag
c
c scan thru all points & check cross product
c
      eps=1e-28
      ixycnt=0
      iyzcnt=0
      ixzcnt=0
      do i=1, n
     if (i.eq.1) then
        d(1) = x(n-1) - x(i)
        d(2) = y(n-1) - y(i)
        d(3) = z(n-1) - z(i)
         else
        d(1) = x(1) - x(i)
        d(2) = y(1) - y(i)
        d(3) = z(1) - z(i)
         endif
     if (i.eq.n) then
        b(1) = x(2) - x(i)
        b(2) = y(2) - y(i)
        b(3) = z(2) - z(i)
         else
        b(1) = x(n) - x(i)
        b(2) = y(n) - y(i)
        b(3) = z(n) - z(i)
     endif

     call acrossb(d,b,a)      ! find unit normal
c
c Division by zero taking place in this section
c was a(c)=a(c)/fmag(a,3)
c
     a(1) = a(1)/(fmag(a,3)+eps)
     a(2) = a(2)/(fmag(a,3)+eps)
     a(3) = a(3)/(fmag(a,3)+eps)
c
c check to see if points reside on x-y plane
c check if axis and unit normal correspond
c
     b(1) = 0     ! axis normal
     b(2) = 0
     b(3) = 1
     call acrossb(a,b,c)
     cmag = fmag(c,3)
     if (cmag.eq.0.0) ixycnt = ixycnt + 1
c
```

```
c check to see if points reside on y-z plane
c
     b(1) = 1    ! axis normal
     b(2) = 0
     b(3) = 0
     call acrossb(a,b,c)
     cmag = fmag(c,3)
     if (cmag.eq.0.0) iyzcnt = iyzcnt + 1
c
c check to see if points reside on x-z plane
c
     b(1) = 0    ! axis normal
     b(2) = 1
     b(3) = 0
     call acrossb(a,b,c)
     cmag = fmag(c,3)
     if (cmag.eq.0.0) ixzcnt = ixzcnt + 1
      enddo
c
c set direction
c
      icnt = 0
      if (ixycnt.eq.n) then
     ipdir = 3 ! surface resides in x-y plane
     planar = 1
     icnt = icnt + 1
      elseif (ixzcnt.eq.n) then
     ipdir = 2 ! surface resides in x-z plane
     planar = 1
     icnt = icnt + 1
      elseif (iyzcnt.eq.n) then
     ipdir = 1 ! surface resides in y-z plane
     planar = 1
     icnt = icnt + 1
      else
     planar = 0
      endif
c
c check checksum
c
      if (icnt.gt.1) then
     write(6,*) 'ERROR: In PLANE routine'
     write(6,*) 'Possible grid problem'
     stop
      endif
      return
      end
c ------------------------------------------------------
c This subroutine finds c = a X b
c a,b,c are 3D vectors
      subroutine acrossb(a,b,c)
      implicit double precision (a-h,o-z)
      double precision a(3), b(3), c(3)
```

```
      c(1) = a(2)*b(3) - b(2)*a(3)
      c(2) = b(1)*a(3) - a(1)*b(3)
      c(3) = a(1)*b(2) - b(1)*a(2)

      return
      end
c -------------------------------------------------------
      double precision function fmag(array,num)
      integer num
      double precision array(num),sum

      sum = 0.0
      do i=1, num
     sum = sum + array(i)*array(i)
      enddo

      fmag = dsqrt(sum)
      return
      end
c ---------------------------------------------------------
      subroutine polint(xa,ya,n,x,y,dy)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      parameter(nmax=dnstr)
      dimension xa(nmax),ya(nmax),c(nmax),d(nmax)
      ns=1
      dif=abs(x-xa(1))
      do i=1,n
         write(101,*) ' polint ',i,xa(i),ya(i)
      enddo
      do i=1,n
        dift=abs(x-xa(i))
        if(dift.lt.dif) then
           ns=i
           dif=dift
        endif
        c(i)=ya(i)
        d(i)=ya(i)
      enddo
      y=ya(ns)
      ns=ns-1
      do m=1,n-1
        do i=1,n-m
          ho=xa(i)-x
          hp=xa(i+m)-x
          w=c(i+1)-d(i)
          den=ho-hp
          if(den.eq.0.) then
             write(6,*) 'polint problem '
             write(6,*)  x
             do ii=1,n
             write(6,*) ii,xa(ii),ya(ii)
             enddo
             stop
```

```
          endif
          den=w/den
          d(i)=hp*den
          c(i)=ho*den
        enddo
        if(2*ns.lt.n-m) then
             dy=c(ns+1)
        else
           dy=d(ns)
           ns=ns-1
        endif
        y=y+dy
       enddo
       return
       end
c
      subroutine plane(ipdir,planar)
      implicit double precision (a-h,o-z)
      include 'csdcfd.par'
      common /str/ x(dnstr),y(dnstr),z(dnstr),ft(dnstr+3),nspts
      integer ipdir, planar
      dimension a(3), b(3), c(3), d(3)
C ------------------------------------------------------------------------
C This routine checks to see if the surface resides within a single plane
C It uses input structural coordinates
C planar = .true.
C        = .false.
C ------------------------------------------------------------------------

c
c external functions
c
      double precision fmag
c
c scan thru all points & check cross product
c
      eps=1e-28
      ixycnt=0
      ixzcnt=0
      iyzcnt=0
      n=nspts
      do i=1, n
     if (i.eq.1) then
        d(1) = x(n-1) - x(i)
        d(2) = y(n-1) - y(i)
        d(3) = z(n-1) - z(i)
          else
        d(1) = x(1) - x(i)
        d(2) = y(1) - y(i)
        d(3) = z(1) - z(i)
          endif
     if (i.eq.n) then
        b(1) = x(2) - x(i)
        b(2) = y(2) - y(i)
```

```
         b(3) = z(2) - z(i)
          else
         b(1) = x(n) - x(i)
         b(2) = y(n) - y(i)
         b(3) = z(n) - z(i)
      endif

      call acrossb(d,b,a)       ! find unit normal
c
c Division by zero taking place in this section
c was a(c)=a(c)/fmag(a,3)
c
      a(1) = a(1)/(fmag(a,3)+eps)
      a(2) = a(2)/(fmag(a,3)+eps)
      a(3) = a(3)/(fmag(a,3)+eps)
c
c check to see if points reside on x-y plane
c check if axis and unit normal correspond
c
      b(1) = 0     ! axis normal
      b(2) = 0
      b(3) = 1
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixycnt = ixycnt + 1
c
c check to see if points reside on y-z plane
c
      b(1) = 1     ! axis normal
      b(2) = 0
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) iyzcnt = iyzcnt + 1
c
c check to see if points reside on x-z plane
c
      b(1) = 0     ! axis normal
      b(2) = 1
      b(3) = 0
      call acrossb(a,b,c)
      cmag = fmag(c,3)
      if (cmag.eq.0.0) ixzcnt = ixzcnt + 1
       enddo
c
c set direction
c
       icnt = 0
       write (6,*) ' Plane counts are ',ixycnt,ixzcnt,iyzcnt
       if (ixycnt.eq.n) then
      ipdir = 3 ! surface resides in x-y plane
      planar = 1
      icnt = icnt + 1
       elseif (ixzcnt.eq.n) then
```

```
      ipdir = 2 ! surface resides in x-z plane
      planar = 1
      icnt = icnt + 1
       elseif (iyzcnt.eq.n) then
      ipdir = 1 ! surface resides in y-z plane
      planar = 1
      icnt = icnt + 1
       else
      planar = 0
c             if plane is curved, then the counts won't be equal
c             to n.  But, the other counts should also be equal
c             to 0.  Correct for this
          if(ixycnt.gt.0.and.ixzcnt.eq.0.and.iyzcnt.eq.0) then
            planar=1
            ipdir=3
            icnt=icnt+1
          endif
          if(ixycnt.eq.0.and.ixzcnt.gt.0.and.iyzcnt.eq.0) then
            planar=1
            ipdir=2
            icnt=icnt+1
          endif
          if(ixycnt.eq.0.and.ixzcnt.eq.0.and.iyzcnt.gt.0) then
            planar=1
            ipdir=3
            icnt=icnt+1
          endif
        endif
c
c check checksum
c
        if (icnt.gt.1) then
      write(6,*) 'ERROR: In PLANE routine'
      write(6,*) 'Possible grid problem'
      stop
        endif
        return
        end
```